Reducing adaptive optics latency using many-core processors

David Barr

A Thesis presented for the degree of Engineering Doctorate

July 21, 2017

Science & Technology Facilities Council
UK Astronomy Technology Centre

Submitted to:

Photonics & Quantum Sciences
School of Engineering & Physical Sciences;
Heriot-Watt University
Scotland

Research carried out at:

UK Astronomy Technology Centre
Royal Observatory
Edinburgh
Scotland

Abstract

Atmospheric turbulence reduces the achievable resolution of ground based optical telescopes. Adaptive optics systems attempt to mitigate the impact of this turbulence and are required to update their corrections quickly and deterministically (i.e. in real-time). The technological challenges faced by the future extremely large telescopes (ELTs) and their associated instruments are considerable. A simple extrapolation of current systems to the ELT scale is not sufficient.

My thesis work consisted in the identification and examination of new many-core technologies for accelerating the adaptive optics real-time control loop. I investigated the Mellanox TILE-Gx36 and the Intel Xeon Phi (5110p). The TILE-Gx36 with 4x10 GbE ports and 36 processing cores is a good candidate for fast computation of the wavefront sensor images. The Intel Xeon Phi with 60 processing cores and high memory bandwidth is particularly well suited for the acceleration of the wavefront reconstruction.

Through extensive testing I have shown that the TILE-Gx can provide the performance required for the wavefront processing units of the ELT first light instruments. The Intel Xeon Phi (Knights Corner) while providing good overall performance does not have the required determinism. We believe that the next generation of Xeon Phi (Knights Landing) will provide the necessary determinism and increased performance. In this thesis, we show that by using currently available novel many-core processors it is possible to reach the performance required for ELT instruments.

Acknowledgements

First, I would like to thank Dr Noah Schwartz, my primary supervisor and main supporter of this research. We have gone through endless revisions and quick turn arounds. Without that this thesis might never of been finished. I would also like to thank my secondary supervisor Dr Robert Thomson based at Heriot Watt.

I would also like to thank the UK Astronomy Technology Centre, who have hosted me during my research over the last four years. Who have supported my work and funded many trips to promote this research at many conferences and workshops around the world. In particular I would like to thank Dr Hermine Schnetler and Andy Vick who were always available and have provide invaluable advice though this research.

This work is part funded by the Science and Technology Facilities Council (STFC), grant ST/K003569/1 and the Centre For Instrumentation. I also gratefully acknowledge support for this research from the UK Engineering and Physical Sciences Research Council, under Grant number EP/L01596X/1. A large portion of this work was carried out in collaboration with the Centre for Advanced Instrumentation at Durham University and in particular I would like to thank Dr Alastair Basden and Dr Nigel Dipper who supported my research.

Lastly I would like to thank my many proof readers who have given valuable advice and comments throughout the writing of this thesis as well as all publications that have come from this research. In no particular order, Dr John Barr, Eve Barr and Dr Stewart Williams.

ACADEMIC REGISTRY
Research Thesis Submission

Name*:*			
School/PGI:			
Version: *(i.e. First, Resubmission, Final)*		Degree Sought (Award **and** Subject area)	

Declaration

In accordance with the appropriate regulations I hereby submit my thesis and I declare that:

1) the thesis embodies the results of my own work and has been composed by myself
2) where appropriate, I have made acknowledgement of the work of others and have made reference to work carried out in collaboration with other persons
3) the thesis is the correct version of the thesis for submission and is the same version as any electronic versions submitted*.
4) my thesis for the award referred to, deposited in the Heriot-Watt University Library, should be made available for loan or photocopying and be available via the Institutional Repository, subject to such conditions as the Librarian may require
5) I understand that as a student of the University I am required to abide by the Regulations of the University and to conform to its discipline.

* *Please note that it is the responsibility of the candidate to ensure that the correct version of the thesis is submitted.*

Signature of Candidate*:*		Date:	

Submission

Submitted By *(name in capitals):*	
Signature of Individual Submitting:	
Date Submitted:	

For Completion in the Student Service Centre (SSC)

Received in the SSC by *(name in capitals):*			
Method of Submission *(Handed in to SSC; posted through internal/external mail):*			
*E-thesis Submitted (**mandatory for final theses)***			
Signature:		Date:	

Please note this form should bound into the submitted thesis.

Updated February 2008, November 2008, February 2009, January 2011

Contents

List of Tables

List of Figures

Acronyms

Acronym	Expanded
AO	Adaptive optics
CoG	Centre of gravity
COTS	Commercial off-the-shelf
CPU	Central processing unit
CuRe	Cumulative Reconstructor with domain Decomposition
CuReD	Cumulative Reconstructor
DM	Deformable mirror
DSP	Digital signal processor
ECC	Error checking code
EE	Encircled energy
ELT	Extremely Large Telescope
FPGA	Field programmable gate array
FRIM	FRactal Iterative Method
FTR	Fourier transform reconstructor
GbE	Gigabit Ethernet
GPU	Graphics processing unit
IP	Internet protocol address
LGS	Laser guide stars
MDE	Multi-core development enviroment
MPSS	Manycore platform software stack
MVM	Matrix vector Multiplication
NGS	Natural guide stars
NIC	Network Interface Card
OS	Operating system
PYR	Pyramid

continued ...

Acronym	Expanded
PYR-WFS	Pyramid wavefront sensor
RMS	Root mean squared
RT	Real-time
RTC	Real-time controller or real-time control system
SH	Shack-Hartmann
SH-WFS	Shack-Hartmann wavefront sensor
SMC	System management controller
SSH	Secure Shell
TCP	Transmission control protocol
TWoG	Thresholded centre of gravity
UDP	User datagram protocol
WCoG	Weighted Centre of gravity
WFS	Wavefront sensor
WPU	Wavefront processing unit

List of publications by the candidate

Peer reviewed

- Barr, D., Basden, A., Dipper, N., & Schwartz, N. (2015). Reducing adaptive optics latency using Xeon Phi many-core processors. Monthly Notices of the Royal Astronomical Society, 453(3), 3222-3233.

Non-peer reviewed

- Barr, D., Basden, A., Dipper, N., Schwartz, N., Vick, A., & Schnetler, H. (2014, August). Evaluation of the Xeon phi processor as a technology for the acceleration of real-time control in high-order adaptive optics systems. In *SPIE Astronomical Telescopes+ Instrumentation* (pp. 91484B-91484B). International Society for Optics and Photonics.

- Barr, D., Basden, A., Dipper, N., & Schwartz, N. (2015) Reducing adaptive optics latency using many-core processors. In Adaptive Optics for Extremely Large Telescopes 4 - Conference Proceedings, volume 1, 2015

- Barr, D., Schwartz, N., Vick, A., Coughlan, J., Halsall, R., Basden, A., & Dipper, N. (2016, July). Novel technology for reducing

wavefront image processing latency. In SPIE Astronomical Telescopes+ Instrumentation (pp. 99094P-99094P). International Society for Optics and Photonics.

Chapter 1

Introduction

Contents

1.1 Motivation

Ever since there creation astronomers have been wanting bigger telescopes. Bigger telescopes give two main advantages over their smaller counterparts: increased light gathering power and resolution. The larger the telescope the more light can be collected, allowing fainter objects to be imaged (the area of the light-gathering surface grows proportionally to the diameter squared). In addition, the larger diameter of the telescope the better the resolution (i.e. the ability to distinguish nearby points into their components). The resolution of an optical system is inversely proportional to the diameter of the aperture, in the case of the modern reflecting telescope the diameter of the primary mirror.

However, at some point the limiting factor on observations is not just the size of the telescope but also the medium through which the observations take place. Astronomical observations from the ground are heavily impacted by the Earth's atmosphere. The turbulence in the atmosphere results in the characteristic 'twinkling' stars that are visible to even the naked eye. This turbulence is produced by local index of refraction variations in the atmosphere. These refractive index variations lead to distortions to the incoming wavefront. A flat wavefront arriving at the Earth's atmosphere becomes distorted as it propagates through it. By the time it reaches the ground based telescopes the wavefront will be distorted, leading to poor image quality. The net effect is a reduction of achievable resolution. Adaptive optics (AO) systems[1] designed to reduce this effect are now in use on most major optical observatory.

Now, over 400 years after the invention of the telescope, the next generation of large optical telescopes are in the design and construction phase. These telescopes have been collectively named the Extremely

Large Telescopes (ELT). Three ELTs are planned with primary mirror diameters ranging from 24 to 40 metres. To counteract the effects of the atmosphere as well as optical aberrations caused by imperfection in the telescope and instruments design and/or manufacture, will require a state-of-the-art AO system.

An AO system is composed of three main elements namely the wavefront sensor (WFS), deformable mirror(DM) and the (real-time controller). A WFS, enables the system to measure the incoming wavefront. A DM, distorts its surface (i.e. creating the inverse shape of the incoming wavefront) to correct the wavefront. Typically the WFS is located after the DM (i.e. in close-loop), so that it only measures the residual errors after correction. Finally, the RTC is used to regularly update the shape of the deformable mirror in order to follow the ever-changing perturbations caused by the atmosphere.

The update frequency of AO systems is typically between a few hundred hertz to a few thousand kilohertz, dictated by the Greenwood frequency (see section 2.2.3). In that short time frame, the RTC needs to process the data coming from the WFS(s) and generate the DM commands. The real-time aspect comes from the fact that the corrections need to be applied before any significant changes in atmospheric turbulence. The high computational load (increasing significantly with telescope diameter) and the very short response time required make it extremely computationally demanding.

Current AO RTC typically make use of custom hardware, using devices such as digital signal processors (DSP) and field programmable gate arrays (FPGA). These technologies are able to meet the latency requirements required for the largest telescopes currently in operation

(i.e 10 m class telescopes). However, the technological challenges that the ELTs and the associated instruments bring are considerable. A simple extrapolation of current systems to the ELTs is not sufficient due to system complexities and long development times these technologies bring. In addition, the observational constraints are becoming increasingly stringent and adaptive opticsystems need to be rethought, for example by allowing for faster development, more scalability, upgradability and maintainability.

In this thesis we investigate a selection of commercial off-the-shelf (COTS) computational hardware technologies that can be used for reducing the latency of large AO systems. We focus on many-core technologies which offer a large number of processing cores, and large memory banks that allow the acceleration of the computationally intensive routines that are performed in the AO control loop. In this thesis we investigate a selection of commercial off-the-shelf (COTS) computational hardware technologies that can be used for reducing the latency of large AO systems. We focus on many-core technologies which offer a large number of processing cores, and large memory banks that allow the acceleration of the computationally intensive routines that are performed in the AO control loop.

These technologies are highly promising due to the potential of meeting the stringent requirements of ELT AO, with large degrees of freedom and high update frequencies, without relying heavily on custom designed and rapidly obsolescent hardware. This thesis focuses on two novel hardware solutions: the Mellanox Tile-Gx and the Intel Xeon Phi. Having not been designed specifically for AO, we investigate these previously untested technologies in the context of AO RTC.

1.2 Layout and scope of thesis

The aim of this thesis is to investigate the suitability of COTS computational hardware for use in AO RTC, with an emphasis on ELT-scale telescopes. This work has lead to a peer reviewed journal paper[2] and three conference proceedings[3, 4, 5]. A reminder of the related publications by the author will be given at the beginning of each relevant chapter. This thesis consists of 8 chapters and has the following layout.

Chapter 2 introduces atmospheric turbulence and the associated difficulties of ground based observation. Chapter 3 continues the introductions and gives a basic overview of AO systems. In particular, this chapter describes the function of each of the components of the system. Chapter 4 introduces the general concepts needed for real-time computing and control systems. Software and hardware issues related to real-time systems are discussed in detail. Chapter 5 covers the specifics of the real-time computing needed for AO. It gives an in-depth review of current and planned AO RTC.

The bulk of the novel work undertaken during the EngD studies of the author begins in chapter 6. Chapter 6 covers the experiments undertaken to characterise the Mellanox Tile-Gx, a many-core processing card. This card is particularly suited to processing the data streamed out of the wavefront sensing camera. Chapter 7 contains a discussion of the experiments undertaken to characterise the Intel Xeon Phi, a many-core processing card designed primarily for high-performance computing. This card is particularly well suited to perform the wavefront reconstruction step of the AO control loop due to the many-cores and large memory bandwidth. We mainly focus on large matrix-vector multiplications algorithm. Chapter 8 concludes this thesis with a dis-

cussion on the results presented in the previous chapters, as well as giving an outlook of possible future research.

References

[1] Horace W Babcock. The possibility of compensating astronomical seeing. *Publications of the Astronomical Society of the Pacific*, pages 229–236, 1953.

[2] David Barr, Alastair Basden, Nigel Dipper, and Noah Schwartz. Reducing adaptive optics latency using xeon phi many-core processors. *Monthly Notices of the Royal Astronomical Society*, 453(3):3222–3233, 2015.

[3] David Barr, Alastair Basden, Nigel Dipper, Noah Schwartz, Andy Vick, and Hermine Schnetler. Evaluation of the xeon phi processor as a technology for the acceleration of real-time control in high-order adaptive optics systems. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 91484B–91484B. International Society for Optics and Photonics, 2014.

[4] David Barr, Noah Schwartz, Andy Vick, John Coughlan, Rob Halsall, Alastair Basden, and Nigel Dipper. Novel technology for reducing wavefront image processing latency. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 99094P–99094P. International Society for Optics and Photonics, 2016.

[5] David Barr, Alastair Basden, Nigel Dipper, and Noah Schwartz. Reducing adaptive optics latency using many-core processors. *Proc. AO4ELT4*, 1, 2015.

Chapter 2

Optical effect of atmospheric turbulence

Contents

Atmospheric turbulence introduces optical distortions to light as it propagates through the atmosphere. Atmospheric turbulence is caused by thermal fluctuations and variations which lead to fluctuations in the density and therefore variations in the refractive index. In turn, this leads to poor imaging quality of optical systems observing the night sky.

In this chapter, we introduce the basic concepts necessary to the understanding of the rest of the work presented. First, we present a description of atmospheric turbulence in section 2.1. We move on to present a statistical description of the atmospheric turbulence in section 2.2. In this section we introduce the major parameters derived through statistical models of the atmosphere. In section 2.3, we discuss the effects of atmospheric turbulence on imaging. In section 2.4, we discuss techniques to mitigate the effect of atmospheric turbulence, and finally we conclude the chapter in section 2.5.

2.1 Atmospheric turbulence

2.1.1 Observing through the atmosphere in popular culture

The effects of atmospheric turbulence causing optical distortions on light coming from stars has been known for centuries. These effects have been captured in academic texts, in popular culture and by artists. Van Gogh's 1889 painting 'Starry Night', gives a partially accurate representation of the swirling turbulence. It also features in a childs nursery rhyme, 'Twinkle Twinkle Little Star' by Jane Taylor, a 19th century poet. The twinkling in the nursery rhyme refers to a process called scintillation, which causes a point source, such as a distant star

to twinkle. This effect is more pronounced near the horizon than at the zenith (the point in the sky directly above an observer) due to the difference in atmospheric thickness which light needs to travel through. In the world of astronomy, Sir Isaac Newton in 1704[1] wrote about the problem of atmospheric turbulence in Opticks.

> *"If the Theory of making Telescopes could be at length be fully brought into practice, yet there would be certain Bounds beyond which Telescopes could not preform. For the Air which we look upon the Stars, is in perpetual Tremor."*

These optical distortions cause a problem for ground based telescopes, reducing the achievable resolution of the images produced. The best possible images are limited by diffraction, hence the term *diffraction limited* images. The achievable resolution can be calculated by the Rayleigh criterion (see equation (2.7)), which suggests the larger the telescopes primary mirror the higher the resolution.

Atmospheric turbulence introduces aberration and further degrades the quality of the produced images. The term *seeing limited* is then used when the resolution is limited by atmospheric turbulence (and not diffraction). This means that without any correction we are unable to recover the ultimate resolution of the telescope. For example in good observation conditions, seeing limited images from a 40 m telescope, without adaptive optics (AO), will have the same resolution as a 10 to 20 cm telescope. Though the large telescope will still benefit from the much larger light gathering capabilities.

2.1.2 Description of turbulence

The atmosphere is not a static system. Movements of the air mass in the atmospheric volume create turbulent eddies. Slight temperature variations within the atmosphere lead to changes in densities, which in turn cause the random mixing of air with local variations in the index of refraction. The index variations along the optical path shifts rays relative to the neighbouring rays. This leads to the gradual, time-dependent deformation of the wavefront. A typical method for quantifying the flow patterns of a fluid is by using the Reynolds number[2]. The Reynolds number (Re) is given by equation (2.1), where V is the velocity of the fluid, L is the scale length of the eddies and v is the kinematic viscosity. The typical scale length of the atmosphere is metres to hundreds of metres, the viscosity is of the order $\approx 1.5 \times 10^{-5}$ m^{2}s^{-1} and if we assume a velocity of a few metres per second we can estimate a Reynolds number of $Re \gtrsim 10^6$. For Reynolds numbers of this magnitude, turbulence is almost always present.

$$Re = V\frac{L}{v} \tag{2.1}$$

When a Reynolds number is significantly large ($Re \gtrsim 4000$), the turbulence will break down. The larger eddies will break down into smaller structures. Eddies break down into smaller and smaller eddies until energy is dissipated at scale lengths ℓ_0. This is known as the Kolmogorov turbulence cascade. The atmosphere will show turbulence with scales varying between ℓ_0 the inner scale and L_0 the outer scale. The inner scale length is typically of the order of a few millimetres while the outer scale length is typically between 10 and 100 metres[3, 4, 5].

2.2 Statistical characterisation of atmospheric turbulence

2.2.1 Index of refraction fluctuations

Due to the stochastic nature of atmospheric turbulence, a statistical approach for describing the turbulence is necessary. A convenient way to describe the spatial structure of a random process is by using the structure function. For a given variable x, measured at points r_1 and r_2 in space where $< . >$ is the statistical average, a structure function can be defined. The structure function $D_x(r_1, r_2)$ is given by equation (2.2).

$$D_x(r_1, r_2) = \left\langle |x(r_1) - x(r_2)|^2 \right\rangle \tag{2.2}$$

The temperature fluctuations cause small variations in the refractive index of air. The structure function that describes temperature and refractive index variations in the atmosphere follows the same law. Assuming that Δ_n is locally stationary, the structure function will only depend on the distance Δ_r between the points. The structure function will follow the Obukhov law[6],

$$D_T(\Delta_r, h) = C_T^2(h)\Delta r^{\frac{2}{3}} \tag{2.3}$$

$$D_n(\Delta_r, h) = C_n^2(h)\Delta r^{\frac{2}{3}} \tag{2.4}$$

Where $C_n^2(h)$ is the structure constant of the refraction index and is expressed in $m^{-\frac{2}{3}}$. It characterises statistically the turbulence strength at altitude h. The structure function of the refractive index is shown in equation (2.4) and shows that these variations are described by the

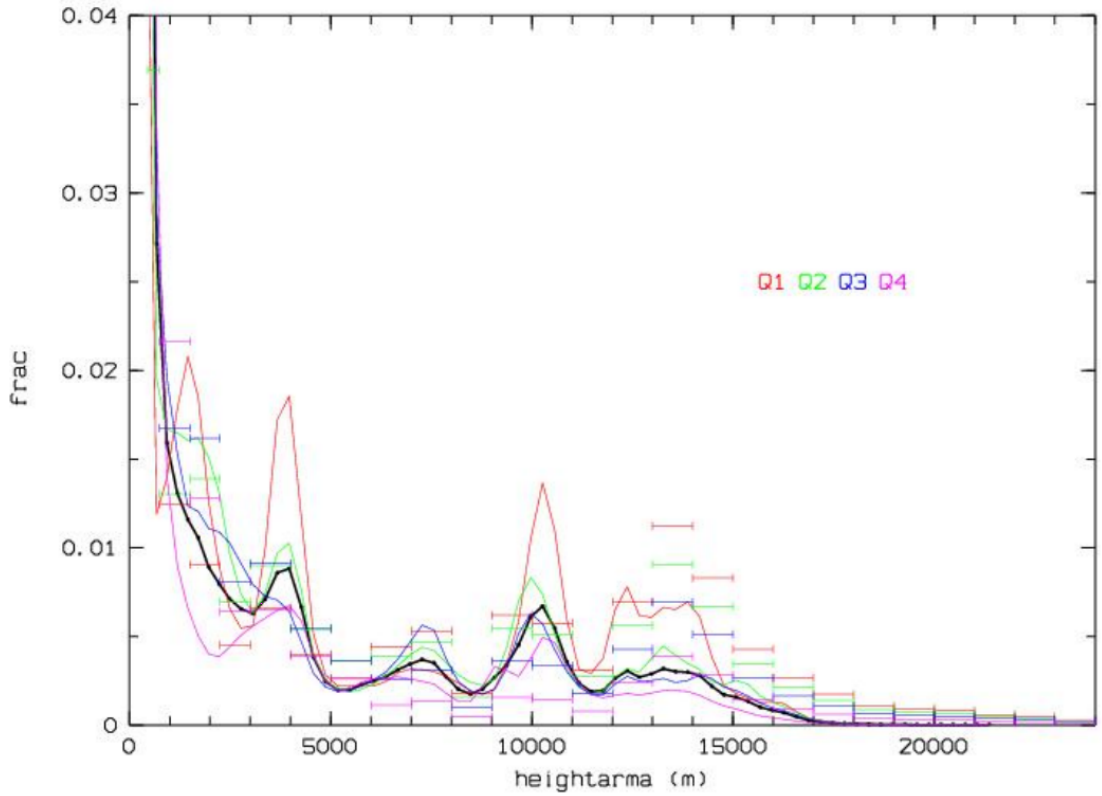


Figure 2.1: Fractional C_n^2 profile at Paranal in 2007 collected using SCIDAR in black compared to models[7].

C_n^2 profile. The C_n^2 values are related to the temperature T, the temperature structure function C_T^2 and atmospheric pressure P, as seen in equation (2.5).

$$C_n^2 = (8.0 \times 10^{-5} \frac{P}{T^2})^2 C_T^2 \tag{2.5}$$

The C_n^2 profile gives the turbulence strength at the different altitudes. The atmospheric turbulence is not uniform and is stratified horizontally in layers. Figure 2.1 shows a typical C_n^2 profile, taken at Paranal, the site of the Very Large Telescopes (VLT), using a SCIDAR. The X axis represents the altitude of the turbulence layer and the Y axis shows the fraction of the C_n^2. Here we see that most of the turbulence is located in the lower altitudes of the atmosphere.

2.2.2 Impact of seeing: the Fried parameter

An important parameter used to characterise atmospheric turbulence is the Fried parameter (r_0)[8].

$$r_0 = [0.423(\frac{2\pi}{\lambda})^2 \frac{1}{\cos\gamma} \int_0^\infty C_n^2(h)dh]^{-\frac{3}{5}} \tag{2.6}$$

Where C_n^2 is the structure constant of the index of refraction, λ the imaging wavelength, h the altitude and γ the zenith angle.

The Fried parameter gives information on the integrated turbulence strength along the line of sight. It is defined as the maximal aperture over which turbulence will not affect the image quality and it is possible to obtain a diffraction limited image. Telescopes larger than r_0 will be limited by turbulence (i.e seeing limited). It can also be seen as the size of a circular aperture over which the root mean square of the phase distortions is approximately equal to 1 rad.

The larger the value of r_0, the 'better' the seeing conditions. It is generally defined for a wavelength of 500 nm and typically ranges between 10 to 30 cm in good conditions. Since r_0 is proportional to $\lambda^{\frac{6}{5}}$, imaging at longer wavelengths is easier.

When imaging an object with an aberration free system, it is possible to obtain diffraction limited images. The angular resolution θ achievable for a telescope with a primary mirror of diameter D at wavelength λ, can be calculated from the Rayleigh criterion given in equation (2.7).

$$\theta \approx 1.22\frac{\lambda}{D} \tag{2.7}$$

In the presence of atmospheric turbulence, the maximum resolution that can be reached is reduced. The angular resolution achievable for

a seeing limited system is given in equation (2.8).

$$d \approx 0.98\frac{\lambda}{r_0} \tag{2.8}$$

Generally, $D >> r_0$ thus implying that the spatial resolution is limited by the seeing rather than diffraction. To get the best scientific result from large ground based telescopes, a form of image correction will need to be applied, typically through AO.

2.2.3 Temporal properties of turbulence

So far in this chapter we have discussed how the spatial properties of atmospheric turbulence affect the incoming light. We have defined a parameter r_0 that characterises how the images are affected by turbulence. As we have defined these terms, we have discussed that the atmosphere is a fluid that is in constant motion.

These temporal properties are important considerations for any AO system, as it is required to make the desired corrections before the atmosphere causes a change in the observed aberrations. The frequency bandwidth of the AO control system that is required to minimise the temporal error is referred to as the Greenwood frequency[9]. The Greenwood frequency is derived from the atmosphere coherence time τ_0.

We define the time constant τ_0, characteristic of the temporal fluctuations of the wavefront, over which the temporal error $\sigma^2_{temporal}$ is lower than 1 rad^2 by:

$$\tau_0 = 0.314\frac{r_0}{\overline{v}}, \tag{2.9}$$

where $\overline{v}$ is the average wind speed weighted by the $C_n^2(h)$ profile. The

Greenwood frequency is defined as $F_G = \frac{1}{\tau_0}$ and is equal to:

$$F_G = 3.185\frac{\overline{v}}{r_0}. \tag{2.10}$$

The mean square phase error $\sigma^2_{temporal}$ associated with a pure delay τ in which the phase is measured at time t and corrected at time $t + \tau$ is then equal to:

$$\sigma^2_{temporal} = (\frac{\tau}{\tau_0})^{5/3} \tag{2.11}$$

The Greenwood frequency is a chromatic parameter. It decreases as $\lambda^{-6/5}$, making AO easier at longer wavelengths. Typical values of τ_0 for $\lambda = 500nm$ are between 1 and 10 ms. In practice, the required frequency bandwidth also depends on the number of controlled parameters (low-order aberrations tend to evolve more slowly than high-order aberrations), the desired final performance and the overall delay of the control loop.

2.3 Imaging through turbulence

2.3.1 Performance of optical systems

For a perfect optical system, the image is limited by diffraction. For a circular aperture without any obscuration, the image of a point source (or point spread function (PSF)) is an Airy pattern. Several optical performance criteria can be used (typically using the PSF), such as encircled energy (EE) and Strehl ratio.

EE is a measure of the concentration energy in an optical image. A typical measurement is of the radius of a PSF that contains a certain percentage (typically 50%-80%) of the total energy. For a point source

(e.g., a distant star) the larger this radius the more spread out the light and the lower the image quality.

The Strehl ratio is a standard optical quality metric in astronomy and commonly used to describe the performance of AO systems. The Strehl ratio is described in equation (2.12), and is the ratio of the central peak of the PSF (corrected or not by AO) to a theoretical image limited by diffraction (i.e. the Airy pattern).

$$\text{Strehl Ratio (SR)} \equiv \frac{PSF(0)}{Airy(0)} \tag{2.12}$$

For small aberrations following Maréchal, this can be expressed as equation (2.13),

$$SR = e^{-\sigma_{res}^2} \tag{2.13}$$

where σ_{res}^2 is the residual phase variance expressed in rad^2.

Figure 2.2 gives a visual representation of the appearance of a series of Strehl ratios.

(a) SR = 100% (b) SR = 80% (c) SR = 50% (d) SR = 20%

Figure 2.2: Typical image quality for a selection of Strehl ratios. copyright: Noah Schwartz

2.3.2 Zernike poylnomials

The Zernike polynomials are a series of polynomials that are orthogonal on a unit circle. They give a convenient way to characterise optical

aberrations as the low-order modes represent the classical optical aberration terms (i.e. focus, astigmatism, coma...). These polynomials can be used to describe the strength of aberrations created by atmospheric turbulence.

The Zernike polynomials were described by Richard Noll in his seminal paper in 1976[10]. The Zernike polynomials are defined by[11],

$$\left.\begin{aligned} Z_{evenj} &= \sqrt{n+1}R_n^m(r)\sqrt{2}\cos m\theta \\ Z_{oddj} &= \sqrt{n+1}R_n^m(r)\sqrt{2}\sin m\theta \end{aligned}\right\} \quad m \neq 0$$
$$Z_j = \sqrt{n+1}R_n^0(r), \qquad m = 0 \tag{2.14}$$

Where R_n^m is the radial polynomial defined in equation (2.15). m and n are positive integers, and r is the radial distance $0 \leq r \geq 1$ as this is defined on a unit circle aperture.

$$R_n^m(r) = \sum_{s=0}^{\frac{n-m}{2}} \frac{(-1^s)(n-s)!}{s![\frac{n+m}{2}-s]![\frac{n-m}{2}-s]!} r^{n-2s} \tag{2.15}$$

From these equations, the Zernike polynomials can quantify any arbitrary wavefront aberrations over a circular aperture. To calculate the wavefront, we can add the contribution from an infinite series Zernike polynomials as in equation (2.16).

$$\varphi(r,\theta) = \sum_j a_j Z_j(r,\theta) \tag{2.16}$$

Where a_j are the coefficients that define the contribution (or strength) of that Zernike mode to the overall phase. Figure 2.3 shows a representation of the first twenty one Zernike polynomials.

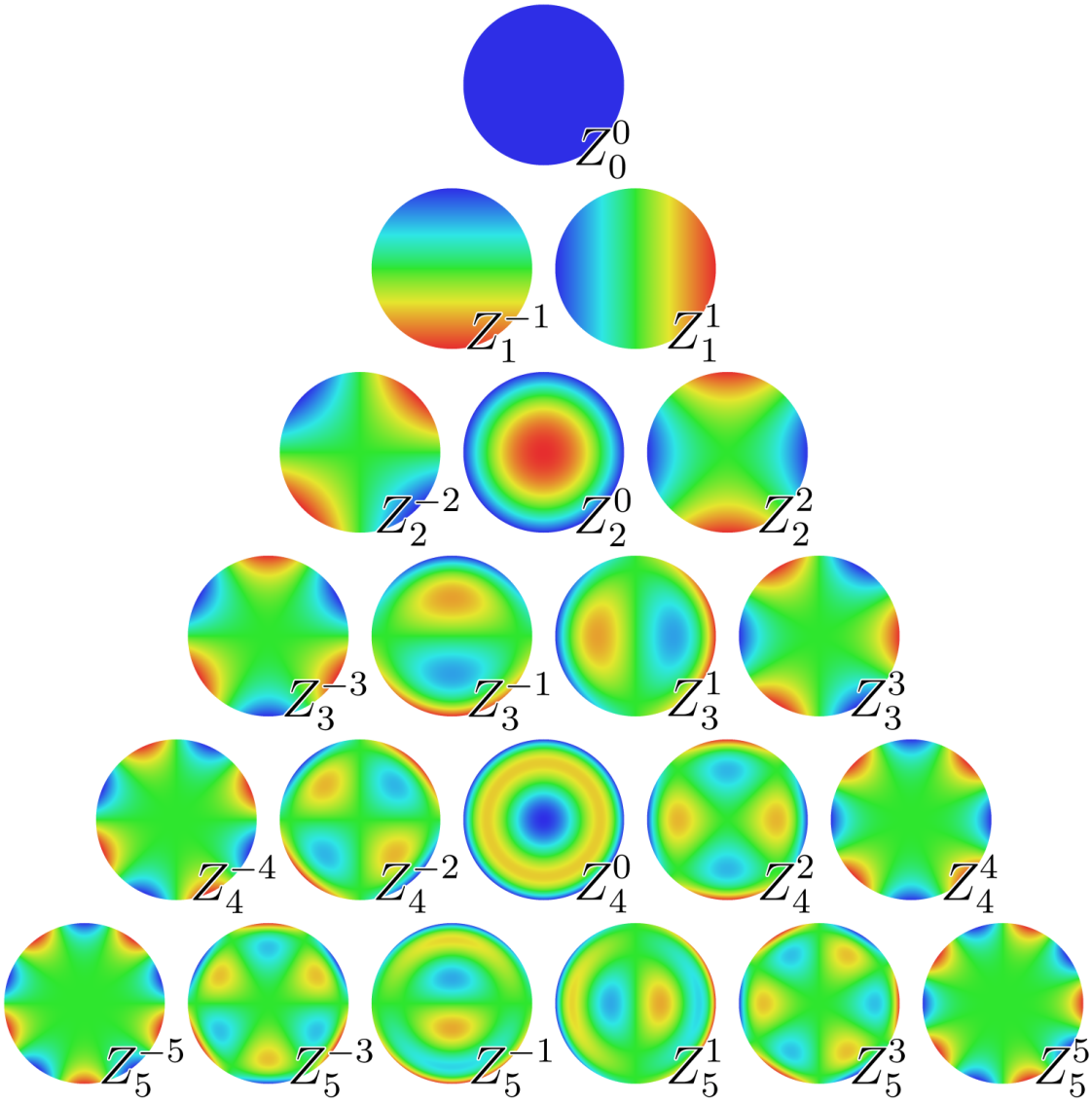


Figure 2.3: Zernike polynomials represented on a unit circle, image copyright Amir Tahmasbi[12]

Following Noll[10], the remaining root mean square (RMS) phase error after J Zernike modes have been perfectly corrected can be given by:

$$\sigma_J^2 \approx 0.29 J^{-\frac{\sqrt{3}}{2}} \left(\frac{D}{r_0}\right)^{\frac{5}{3}} \tag{2.17}$$

Correcting for tip and tilt only, enables the system to remove approximately 85 % of the wavefront variance due to atmospheric turbulence[13]. Because of the large variation in contributions, the lower order Zernike

are sometimes corrected using a dedicated tip-tilt mirror.

Zernikes are not the only phase decomposition basis that can be used to describe the wavefront. Karhunen-Loève modes are also a good example[14, 15], but don't have an analytical expression for atmospheric turbulence. Other bases such as the deformable mirror (DM) modes of an AO system can also be used for example.

2.4 Mitigating the impact of atmospheric turbulence

There are several different imaging techniques designed to compensate for the atmospheric turbulence, and can provide better image quality than seeing-limited imaging.

2.4.1 Speckle imaging

A first solution to overcome the problem of the turbulent atmosphere is speckle imaging[8]. This is where the final image is created by using multiple short exposures. During a short exposure, it can be assumed that the atmosphere is static and distorting effects are limited to the high-order aberrations. The final image is built by taking lots of short exposure images and stacking them according to the brightest spot or centroid (i.e. shift and add method). The major advantage of this method is to partially mitigate tip-tilt aberrations. However, this method is still very limited in its signal-to-noise improvement in the final image.

2.4.2 Lucky imaging

Proposed by David Fried in 1978[16] lucky imaging is similar to speckle imaging. It uses a series of short-exposure images that are co-added using a 'shift-and-add' method. Where this method differs from the previous one is that not all images are used, and only the best images are added together. The atmosphere is not constant from one frame to another, and for some frames the atmosphere causes less distortions than during others. Typically, 1 % of frame will be used in the final images[17], depending on the atmospheric turbulence and resolutions required.

Technology evolution (cheap video and web cameras) has allowed many amateur astronomers to use this method to capture images at much higher resolution than ever before and on a modest budget.

2.4.3 Speckle interferometry

Invented by Antoine Labeyrie[18], speckle interferometry is similar to speckle imaging. In speckle interferometry, large numbers of short exposure images are taken. The Fourier transform of each image is then taken to obtain the diffraction pattern. The square modulus of all the diffraction patterns are added together, an average of the entire diffraction pattern is taken, then an inverse Fourier transform is applied. The recovered result is an autocorrelation of a diffraction limited image of the object.

2.4.4 Deconvolution by wavefront analysis

Deconvolution is a post processing technique for image reconstruction [19]. A series of short exposure images are taken and simultaneous,

wavefront data recorded. From the measured phase, an optical transfer function can be calculated. This optical transfer function is deconvolved with the distorted image to try and reconstruct a non-distorted image.

2.5 Conclusion

The atmosphere is in perpetual movement. This turbulence creates index of refraction variations that in turn introduce optical distortions to the light traveling through the atmosphere. In this chapter, some of the basic atmospheric turbulence properties were introduced. We have shown how it impacts the formation of images for ground based astronomy. We also briefly described some mitigation techniques.

None of the presented methods so far enable a real-time compensation of atmospheric turbulence. This means that they will be unable to fully retrieve the full diffraction-limited capability of the telescope. The final image will therefore have a limited signal-to-noise ratio. AO, which is a real-time technique to mitigate time-varying aberrations, is presented in the next chapter.

References

[1] Isaac Newton. *Opticks, or, a treatise of the reflections, refractions, inflections & colours of light*. Courier Corporation, 1979 (Originally 1704).

[2] Osborne Reynolds. An experimental investigation of the circumstances which determine whether the motion of water shall be direct or sinuous, and of the law of resistance in parallel chan-

nels. *Proceedings of the royal society of London*, 35(224-226):84–99, 1883.

[3] DF Buscher, JT Armstrong, CA Hummel, A Quirrenbach, D Mozurkewich, KJ Johnston, CS Denison, MM Colavita, and M Shao. Interferometric seeing measurements on Mt. Wilson: power spectra and outer scales. *Applied Optics*, 34(6):1081–1096, 1995.

[4] J Davis, PR Lawson, AJ Booth, WJ Tango, and ED Thorvaldson. Atmospheric path variations for baselines up to 80m measured with the Sydney University Stellar Interferometer. *Monthly Notices of the Royal Astronomical Society*, 273(1):L53–L58, 1995.

[5] Rodolphe Conan, Aziz Ziad, Julien Borgnino, Francois Martin, and Andrei A Tokovinin. Measurements of the wavefront outer scale at paranal: influence of this parameter in interferometry. In *Astronomical Telescopes and Instrumentation*, pages 963–973. International Society for Optics and Photonics, 2000.

[6] AM Obukhov. Structure of the temperature field in a turbulent current. *Izv.Aka. Nauk SSSR, Ser.*, 13, 1949.

[7] Marc Sarazin, Miska Le Louarn, Joana Ascenso, Giancluca Lombardi, and Julio Navarrete. Defining reference turbulence profiles for E-ELT AO performance simulations. In *Third AO4ELT Conference Adaptive Optics for ELTs Proc., Edited by S. Esposito and L. Fini*, 2013.

[8] D. L. Fried. Optical resolution through a randomly inhomogeneous medium for very long and very short exposures. *J. Opt. Soc. Am.*, 56(10):1372–1379, Oct 1966.

[9] Darryl P Greenwood. Bandwidth specification for adaptive optics systems. *JOSA*, 67(3):390–393, 1977.

[10] Robert J Noll. Zernike polynomials and atmospheric turbulence. *JOsA*, 66(3):207–211, 1976.

[11] Virendra N Mahajan. Zernike circle polynomials and optical aberrations of systems with circular pupils. *Applied optics*, 33(34):8121–8124, 1994.

[12] Amir Tahmasbi. Amir tahmasbi - zernike, 2016. "Accessed: 2016-08-13".

[13] Joseph Winocur. Modal compensation of atmospheric turbulence induced wave front aberrations. *Applied optics*, 21(3):433–438, 1982.

[14] Nicolas A Roddier. Atmospheric wavefront simulation using zernike polynomials. *Optical Engineering*, 29(10):1174–1180, 1990.

[15] Michael C Roggemann, Byron M Welsh, and Bobby R Hunt. *Imaging through turbulence*. CRC press, 1996.

[16] David L Fried. Probability of getting a lucky short-exposure image through turbulence. *JOSA*, 68(12):1651–1658, 1978.

[17] Nicholas M Law, Craig D Mackay, and John E Baldwin. Lucky imaging: high angular resolution imaging in the visible from the ground. *Astronomy & Astrophysics*, 446(2):739–745, 2006.

[18] Antoine Labeyrie. Attainment of diffraction limited resolution in large telescopes by fourier analysing speckle patterns in star images. *Astron. Astrophys*, 6(1):85–87, 1970.

[19] J Primot, G Rousset, and JC Fontanella. Deconvolution from wave-front sensing: a new technique for compensating turbulence-degraded images. *JOSA A*, 7(9):1598–1608, 1990.

Chapter 3

Adaptive Optics

Contents

In this section we describe, from a theoretical point of view, the main elements constitutive and fundamental limitations of Adaptive Optics (AO). The objective of this chapter is to introduce the necessary formalism in the case of classical AO, introduce its limitations and briefly present some mitigation and solutions with techniques such as wide-field AO. Our goal is not to explain the detailed operation performed by the AO system, but to give a basic understanding of the process at play, with a focus on the temporal and fitting errors. These error terms and expanding the current AO systems to the extremely large telescopes (ELTs) are the primary motivation driving the research in this thesis.

3.1 Historical aspects of adaptive optics

AO was first proposed in 1953 by Horace Babcock to compensate for atmospheric turbulence in real-time[1]. At the time the idea could not be realised (or implemented) due to a lack of suitable technologies and therefore remained a concept. AO was first developed by the American defence industry in the 1970s to image satellites from the ground. The Defense Advancement Research Projects Agency (DARPA) awarded one year contracts to three companies (Itek, Perkin Elmer and Hughes Research Centre). The competition carried on for another few years and eventually led to the development of Rayleigh guide stars[2].

It was not until decades later in 1990, that work began on developing the first AO system for astronomy. The first major application of AO within astronomy was a project called COME-ON[3]. It was installed at the European Southern Observatory 3.6 m telescope, at the time one of the largest telescopes in the world. The results of the project

were promising, achieving Strehl ratios of up to 0.3 at 2.2 μm under average seeing conditions (in the order of 0.8 arcseconds)[4]. Due to the promising nature of the results, COME-ON was upgraded COME-ON+ in 1992. It then entered its third stage called Adaptive Optics, Near Infrared System (ADONIS).

The ADONIS AO system was composed of a 32 sub-aperture (on a 7×7 grid) Shack-Hartmann wavefront camera (using a 64×64 pixel CCD detector) combined with a deformable mirror (DM) with 52 actuators[5]. A dedicated real-time controller (RTC) was developed for the project, an architecture relying on VME motherboards and dedicated DSP C40 modules. The RTC was split into 2 modules, a wavefront computer dedicated to the wavefront sensor data reduction (windowing, flat-fielding and thresholding) and a command computer (determining the wavefront slopes in x and y and computing the mirror commands). The master computer (a Linux workstation) was interfaced to each of the AO system elements by means of Ethernet or RS-232 links. Overall the maximal practical DM command rate was 100 Hz.

Since the COME-ON project, AO has become an important addition to most large ground based telescopes.

3.2 Principle of AO

The aim of astronomical AO systems is to compensate for distortions created by atmospheric turbulence. AO is an opto-mechanical system, heavily reliant on computations that corrects in *real-time* the incoming turbulent wavefront that has been intercepted by the telescope. The received wavefront distortions are measured using a wavefront sensor . From these WFS measurements, a deformable mirror is used to obtain

a flattened wavefront that enables the instrument to observe (in the imaging path) a corrected image near the theoretical limit of the telescope imposed by diffraction. In order to achieve this, the AO systems rely on the three key elements:

- A wavefront sensor that measures the incoming wavefronts.
- A wavefront corrector (or deformable mirror, DM) that ensures the correction by adding a deformation to the incoming wavefront that is inverse to the one measured by the WFS.
- A real-time controller that processes the WFS measurements in real-time in order to control the DM.

A simplified version of a closed-loop AO system is shown in figure 3.1. The distorted light enters the telescope and is reflected off the deformable mirror (in a closed-loop configuration). The light is split, with the majority of the light going to the science camera or instrument, and a small amount of light going to a wavefront camera.

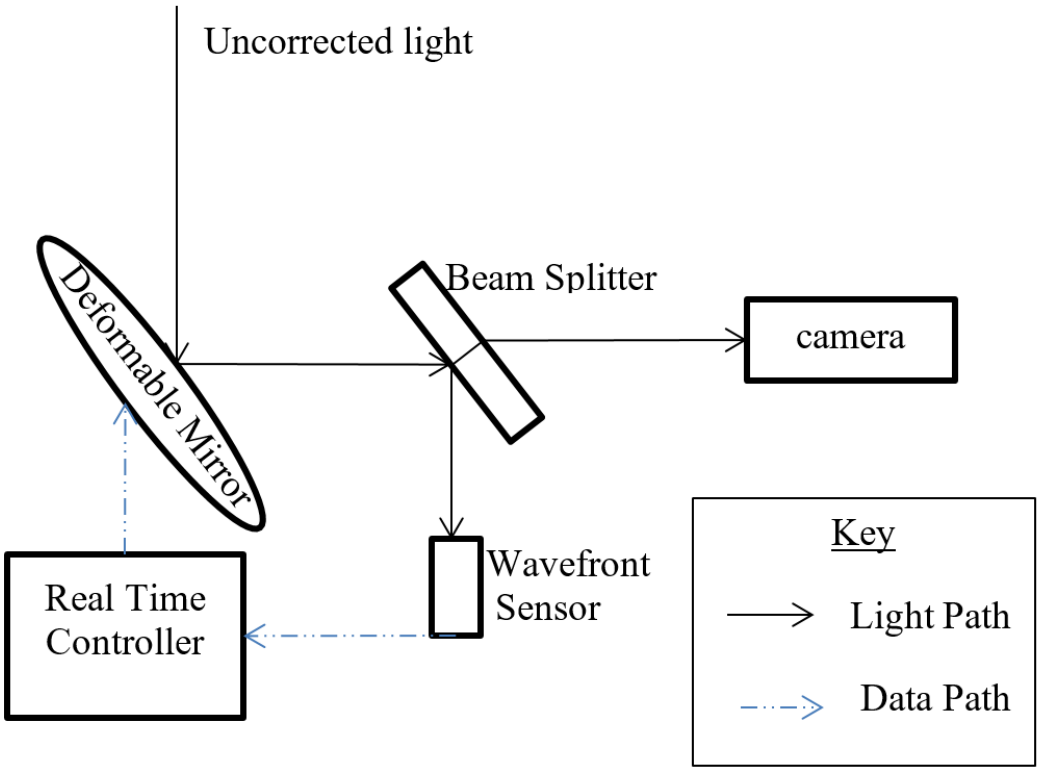


Figure 3.1: Simplified diagram of a closed-loop adaptive optics system.

3.3 Wavefront sensing

To be able to correct the aberrations caused by the atmospheric turbulence, the system has to be able to sense these aberrations present in the wavefront. This is done with a device called a wavefront sensor (WFS). Simply put, the WFS can be seen as the 'eyes' of an AO system: the WFS senses the distortions of the wavefront and sends this information to the control computer for the deformable mirror commands to be calculated. There are many different types of WFSs. The two WFSs that are mainly used in astronomical AO are the Shack-Hartmann and the Pyramid. Other types of WFSs exist (such as curvature sensors), however these will not be covered in this thesis.

3.3.1 Shack-Hartmann Wavefront Sensors

The Shack-Hartmann Wavefront Sensor (SH-WFS) was created in the 1960s by the US military when trying to image satellites[6]. The SH-WFS is an improvement of the Hartmann test; which was used to compute the aberrations present in the incoming light and to test optical systems. The Hartmann test did this by placing in the light path a metal disk with a regular grid of holes. If no aberrations were present in the system, the spots created by the Hartmann screen form a square array (since the screen itself is a square array of holes). However, if aberrations are present, they can be determined by measuring the spots at a series of locations. This worked well when measuring large low-order aberrations but performed poorly for small amplitude, high-order aberrations. As atmospheric turbulence has both large low-order aberrations as well as low amplitude high-order aberrations, the Hartmann test is not ideal for AO. To create the Shack-Hartmann WFS, the holes on the Hartmann test were replaced with a lenslet array. This gives better coverage, and therefore there are fewer wasted photons and more sensitivity to higher order aberrations.

The SH-WFS is a 2D lenslet array that focuses incoming light onto a detector, as shown in figure 3.2. Each lenslet focuses the incoming light onto a small area of pixels called a sub-aperture. If the incoming wavefront is flat and parallel to a lenslet, the light will be focused at the centre of that sub-aperture. If the wavefront is not locally parallel to the lenslet, the light will not focus at the centre of that sub-aperture. The SH-WFS is the most common WFS used in AO systems. It has a simple premise and is generally simple to operate.

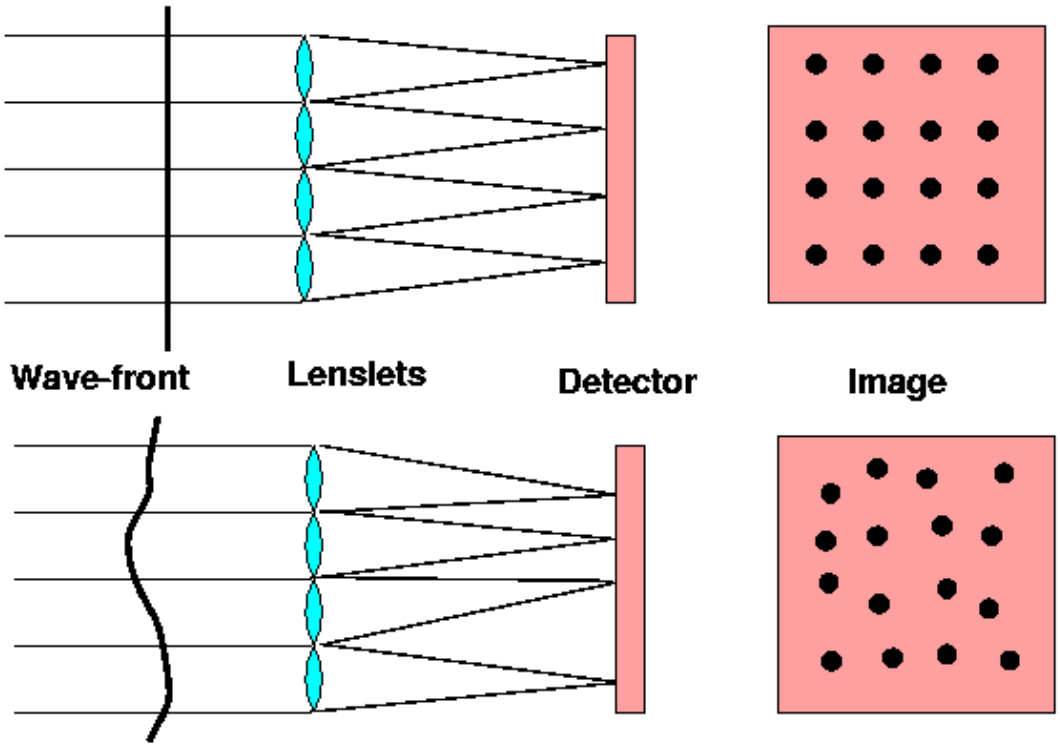


Figure 3.2: Schematic diagram of a Shack-Hartmann wavefront sensor. Copyright: CTIO

From the displacement of the spot relative to the centre of that sub-aperture, the local slope of the wavefront can be calculated in both the X and Y direction. In practice, and in order to calibrate precisely the focus positions of the lenslet array, we measure the actual position of the spots on the detector using a reference source in the instrument (see top part of figure 3.2)[7]. The recorded wavefront will then give you references, that are called the reference slopes. They can also contain an additional offset to take into account, for example, differential errors arising between the sensing and imaging paths (i.e. non-common path aberrations). The location of the spots are typically evaluated by calculating the centre of gravity of the focal spot falling on the sub-apertures.

The SH-WFS measures the mean gradient of the phase $\bigtriangledown\phi(x, y)$ over

each sub-aperture and can be defined as:

$$\bigtriangledown\phi \equiv \begin{pmatrix} \frac{\partial}{\partial x}\phi \\ \frac{\partial}{\partial y}\phi \end{pmatrix} \tag{3.1}$$

The local gradients displace the centre of the image at the focus of the sub-apertures according to:

$$\begin{aligned} \frac{\Delta x}{f_0} &\equiv \frac{\lambda}{2\pi\Omega_p} \int\int_{\Omega_p} \frac{\partial\phi(x,y)}{\partial x} \partial x \partial y \\ \frac{\Delta y}{f_0} &\equiv \frac{\lambda}{2\pi\Omega_p} \int\int_{\Omega_p} \frac{\partial\phi(x,y)}{\partial y} \partial x \partial y \end{aligned} \tag{3.2}$$

where f_0 is the lenslet focal length and Ω is the surface area. Equation (3.1) can be written in matrix form: $s = D\Phi + n$, where s is the measurement vector of the local gradients, ϕ the phase, D the matrix giving the relationship between the gradients of a sub-aperture as a function of the phase, and n is the measurement noise.

One of the important limitations of the SH-WFS is that it measures the local slope of the incoming wavefront and is therefore unable to measure piston[1]. If the incoming light has residual piston, the spots will remain in the same location. For this reason, waffle mode[2], where the incoming wavefront is composed of a checkerboard pattern matching the lenslet geometry, cannot be measured by the Shack-Hartmann WFS.

3.3.2 Pyramid Wavefront Sensors

Roberto Ragazzoni first suggested in 1995, the pyramid wavefront sensor (PYR-WFS) as an alternative to the SH-WFS[8]. The PYR-WFS is a modified version of the Foucault knife-edge test[9], which is still used by many amateur telescope makers. The incoming light is focused

[1]piston refers to the first Zernike mode, which is a flat wavefront but moved in phase.

[2]Waffle is a form of wavefront where each lenslet sees a flat wavefront, although each are out of phase with one another.

on the apex of a square based pyramid, thus creating 4 pupil images onto a detector. A schematic diagram of a PYR-WFS is presented in figure 3.3. The phase derivative in x and y are related to the intensity difference on each of the 4 quadrants according to equation (3.3). A small modulation of the PSF position hitting the apex of the pyramid can be added, generally introduced by a fast tip-tilt mirror, in order to increase the linearity range of the measurement.

This type of WFS will not be studied in detail in this thesis. However, algorithmic complexity for the PYR-WFS and SH-WFS are very similar and results obtained with a SH-WFS can be generalised to the PYR-WFS.

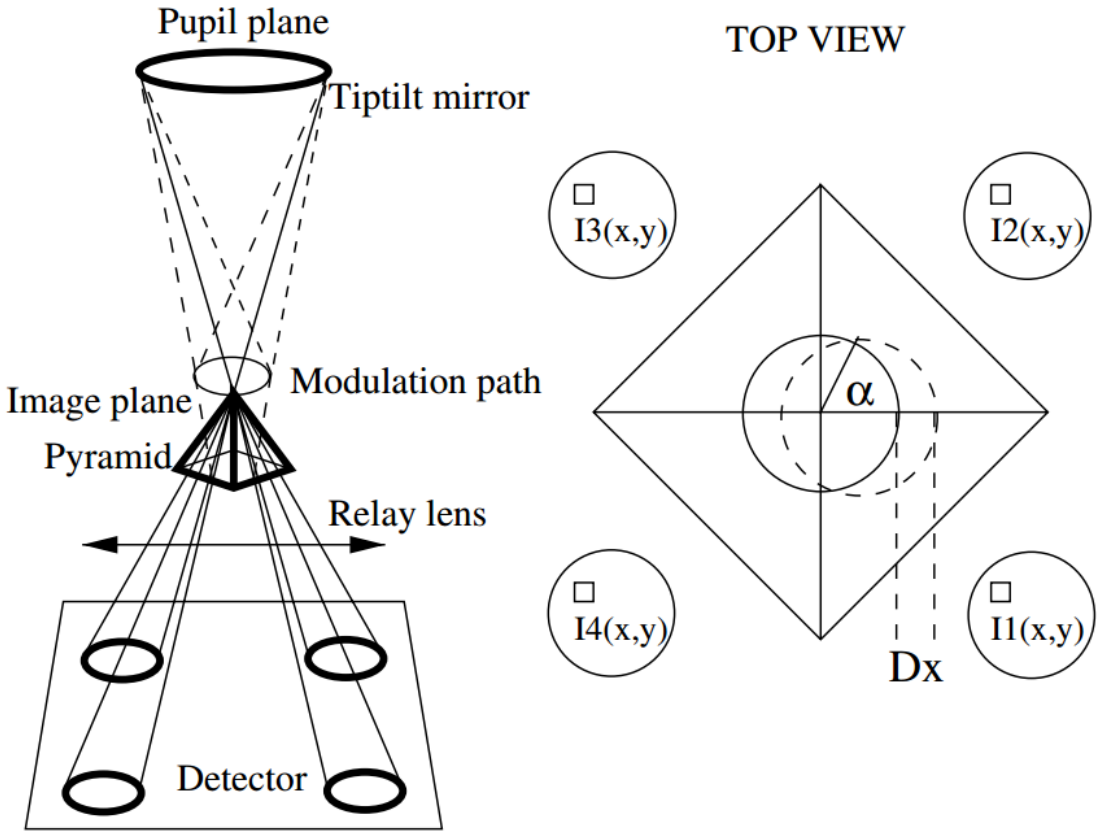


Figure 3.3: Schematic diagram of a Pyramid wavefront sensor. Copyright: Véniraud et al 2004[10]

$$D_x = \frac{I_1(x,y) - I_2(x,y) - I_3(x,y) + I_4(x,y)}{I_1(x,y) + I_2(x,y) + I_3(x,y) + I_4(x,y)}$$
$$D_y = \frac{I_1(x,y) + I_2(x,y) - I_3(x,y) - I_4(x,y)}{I_1(x,y) + I_2(x,y) + I_3(x,y) + I_4(x,y)} \quad (3.3)$$

The PYR-WFS have shown higher sensitivity than SH-WFS in closed-loop systems[11]. This allows for dimmer guide stars to be used than by the SH-WFS. Many first light instruments of the ELTs are planning to use the pyramid WFS as the baseline[12].

3.3.3 Guide Stars

In order to measure the incoming wavefront, a small amount of light is picked off and directed into the WFS; the rest of the light goes to the science camera or instrument (if sufficiently bright, the science object itself can be used). The reference used to measure the wavefront is called a guide star. This is a star in close proximity to the object of interest, and is sufficiently bright for its wavefront to be measured with a high enough signal-to-noise ratio. There are two types of guide stars used, the Natural Guide Stars (NGS) and artificially created Laser Guide Stars (LGS).

NGS have to be close enough to the science target being observed so that the light traveling can be assumed to have been affected by the same atmospheric turbulence (anisplanatism effect). Unfortunately, the number of bright stars in the sky is quite low (typically in the area of 1% of the sky).

3.3.3.1 Laser Guide Stars

Artificial laser guide stars can be used to increase the sky coverage. There are two types of LGS: Rayleigh LGS and Sodium LGS.

Rayleigh scattering LGSs are the simplest to understand and cheaper

to produce. These guide stars are created by the scattering of light from particles smaller than the wavelength of the laser. When the density of particles drops below a certain level, Rayleigh scattering guide stars become too faint to be used. This altitude is typically around 8-12km for certain wavelengths, but can go up to 30km[13].

On the other hand Sodium LGS can go up to 90 km[14] and use a laser at a wavelength of 589.2 nm. The sodium lasers are used to excite sodium atoms within the sodium layer. The sodium atoms absorb this energy and then re-emit light as the atom falls back to its original energy state[15].

The sodium laser guide star is much higher in the atmosphere than the Rayleigh laser guide star. This higher altitude means that the returning light is effected by more atmospheric turbulence, and that the WFS can measure more of the turbulence volume. Sodium LGSs are the baseline for all planned ELT AO systems.

3.3.3.2 Limitations of Laser Guide Stars

LGS are becoming common place on most telescopes. They are used to increase the sky coverage but come with a number of limitations, namely:

- Tip-tilt uncertainty,
- Cone effect,
- Sodium layer fluctuations,
- Spot elongation.

The light of the LGS goes through the atmosphere twice. Once on its way up to the sodium layer and once on its way back. Because

of this double path problem, any tip-tilt error will not be measurable (i.e. tip-tilt uncertainty). Typically, an NGS is used to measure low-order modes and in particular tip-tilt. This can be done using few sub-apertures and therefore on dimer NGSs[16].

The second major limitation of the laser guide star is the cone effect. The star laser has a finite altitude (90 km), and the wave that is received from the laser source is a spherical wave. Therefore, the phase perturbations are expanded from the initial perturbations (this is especially true for the high altitude turbulence). By measuring these expanded aberrations, we add an additional error. The larger the telescope diameter, the larger the cone effect. In addition, the cone of light coming from the LGS will only probe a cone, not the entire column of turbulence that is seen by the science target[17]. This is especially true for the higher altitudes and is shown in figure 3.4.

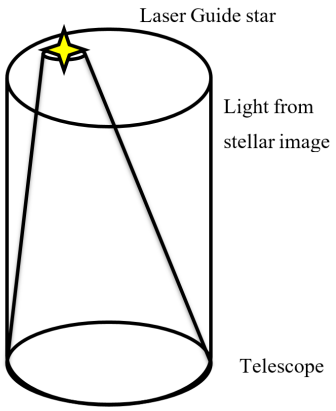


Figure 3.4: Illustration of the cone effect of a laser guide star.

Finally, the third additional error when using lasers results from the 3 dimensional structure of the source. The sodium layer is not static, like the rest of the atmosphere it is constantly changing. Fluctuations in the altitude and structure of the sodium layer can introduce focus errors (but other errors as well) in the AO system. The thickness (approxi-

mately 10 km) of the Sodium layer leads to the LGS being an extended source (i.e. spot elongation). Both these factors contribute to larger wavefront errors degrading the performance of the AO system[18].

3.4 Wavefront correction

To correct the wavefront in the AO system, a corrective element called a deformable mirror is used. In many AO systems the DM (or DMs) can be used in conjunction with a number of steerable mirrors. The lowest order optical modes (i.e. tip and tilt, see section 2.3.2) require more stroke than DMs typically provide. For this reason, a separate fast steerable tip/tilt mirror can be used to remove these low order modes, making it necessary to separate the correction onto multiple distinct devices. Simply put, the DM(s) can be seen as the 'hands' of the AO system.

3.4.1 Deformable mirror

The DM, by changing the shape of its surface, corrects the wavefront aberrations by creating phase changes in the incoming wavefront (i.e. by allowing different parts of the wavefront to travel different distances). A simplified version of how the DM modifies the incoming wavefront is shown in figure 3.5.

It is composed of a reflective surface that is controlled by actuators (or motors) located at the back of this surface. There is a wide range of different technologies suitable for the construction of the DMs. A detailed description of these competing technologies can be found in[19]. The DM, regardless of the manufacturing technology, is characterised by its spatial and temporal properties.

Neglecting temporal dynamics (i.e. infinitely small response time of the DM), we then define a linear relationship between the applied voltage u to the mirror and its deformation by: $\phi_{cor} = Nu$, where N is a matrix containing the DM influence functions and is called the influence matrix, and ϕ_{cor} is the deformation of the correction phase generated by the DM.

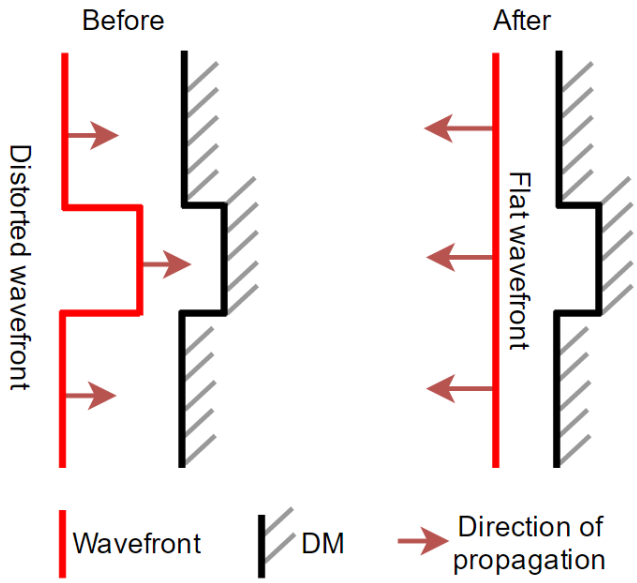


Figure 3.5: Illustration of how the DM modifies the incoming wavefront.

3.4.2 Fitting Error

Due to the technological challenges in developing DMs, the DMs are not perfect and cause some error in the system, called the fitting error. The fitting error is the error caused by the inability of the DM to perfectly replicate the incoming wavefront. One of the important considerations for the fitting error is the number of actuators, generally the higher the number of actuators the smaller the fitting error. This allows the DM to better match the wavefront and especially higher frequency aberrations.

The DM fitting error can be written as:

$$\sigma^2_{fitting} = a_F \left(\frac{d_s}{r_0}\right)^{\frac{5}{3}} \tag{3.4}$$

Where, d_s is the inter-actuator distance, r_0 the Fried parameter and a_F the fitting error coefficient. This coefficient, in particular, depends on the actuator geometry and whether the DM mirror surface is a continuous face-sheet or made of separate segments. For example, a_F is equal to 0.232 for a square geometry and 0.200 for a hexagonal geometry for continuous face-sheets[20].

Another important consideration when selecting the DM is the stroke and inter-stroke of the actuators. The stroke refers to the range of motion each actuator has. Larger strokes allow larger amplitudes to be corrected. The inter-stroke refers to the maximal distance achievable between two neighbouring actuators (i.e. one pulling, one pushing). It is also important to consider the coupling factor between actuators which shows how much the movement of one actuator will displace its neighbors (i.e. cross-coupling).

3.5 Real-time control

The RTC must collect the measurements (pixel data) from various NGS or LGS WFS(s) and drive the DM(s). Simply put, the RTC can be seen as the 'brain' of the AO system.

In simplified terms, the RTC hard real-time pipeline can be expressed in two main steps. The first of these steps is the wavefront processing stage, where the wavefront slope vector is extracted from the incoming pixel stream coming from the WFS(s). The second step is referred to as the wavefront reconstruction or control calculation, where the

DM commands are calculated using the slope vector. Finally, the DM commands are sent to the DM control electronics to be applied.

A selection of algorithms that can be used to perform either the wavefront processing and wavefront reconstruction can be found in section 5.2 and 5.3 respectively.

The control of an AO system is realised in a dynamic way. It must be realised quickly enough to correct atmospheric turbulence in real-time, but it also depends on the temporal characteristics of each of its components. All AO systems can therefore be characterised by its timing diagram (or chronogram). These can prove to be quite complex depending on the architecture of the system and its components. More details on the AO chronogram are presented in section 5.1.2.

Apart from the hard real-time operations, the RTC needs to do a number of additional tasks. These tasks typically include, but are not limited to, supervising operations such as atmospheric turbulence monitoring and sodium layer profiles, handling configurations, calibrations, loop optimisations required by the AO system, and recording of telemetry and diagnostic data. The particularly challenging aspects in the era of the ELTs are the hard real-time functions, and will therefore be the main focus of this work.

3.5.1 Temporal error

The AO control loop introduces a delay in the correction that is added to the delay already introduced by the WFS measurement. The temporal error will increase as the latency of the AO system increases.

Minimising the temporal error drives the systems update frequency: the faster the update frequency, the lower the temporal error. The

mean square phase error $\sigma^2_{temporal}$ associated with a pure delay τ (phase is measured at time t and correction applied at time $t+\tau$) is given by equation (3.5). τ_0 is the Greenwood time delay and $\langle v \rangle$ is the mean transverse turbulence velocity (weighted by the strength of the turbulence layer). The required control frequency of the system is called the Greenwood frequency. Technologies for minimising the time taken to perform the required calculations, and therefore minimising the temporal error term, is the focus of the research presented in this thesis.

$$\sigma^2_{temporal}(\tau) = \left(\frac{\tau}{\tau_0}\right)^{\frac{5}{3}} = \left(\frac{\tau}{0.314\frac{r_0}{\overline{v}}}\right)^{\frac{5}{3}} \tag{3.5}$$

3.6 Intrinsic errors of an AO system

When designing systems such as AO systems, the aim is to reduce the errors in the flattened wavefront. Even so, errors will still be present due to imperfection in manufacturing, technological limitations or cost. There are many areas in the AO systems that can introduce errors, here we highlight below four of the main sources of these errors:

- Errors related to the measurement: σ^2_{noise} and $\sigma^2_{aliasing}$. The WFS realises a spatially sampled measurement of the phase (defined by the number of lenslets). The high spatial frequencies of the turbulence are poorly sampled and will be aliased: the high frequencies fold onto the lower frequencies. By the nature of the measurement itself, detector and photon noise will be present.

- Errors related to the correction: the fitting error, $\sigma^2_{fitting}$ (see section 3.4.2).

- Errors related to the control loop: the temporal error, $\sigma^2_{temporal}$ (see

section 3.5.1).

- Errors related to calibrations: σ^2_{calib}. This take into account the error linked to the measurement of the interaction matrix, of the reference slopes and the use of models in the control law.

This is not an exhaustive list as we have only emphasised some of the main error terms. While we present each error term as being independent, in a real AO system many may be in fact correlated. In addition, many other errors can play a non-negligible role on the final performance depending on the system characteristics and may need to be considered during a comprehensive AO study. For a more complete list of error terms see for example[21].

The temporal error $\sigma^2_{temporal}$ and the fitting error $\sigma^2_{fitting}$ have been introduced in detail in sections 3.5.1 and 3.4.2. Turbulence induced aberrations are also importance and include scintillation and anisoplanatism. The total residual error is the quadratic sum of these errors:

$$\sigma^2_{residual} = \sigma^2_{noise} + \sigma^2_{fitting} + \sigma^2_{temporal} + ... \tag{3.6}$$

Reducing these errors will increase the optical performance and raise the Strehl Ratio achievable with the AO system.

3.7 Adaptive optics concepts

3.7.1 Closed-loop and open-loop AO

A closed-loop architecture is the preferred layout for many AO instruments. As shown in the block diagram (Figure 3.6), the wavefront is measured after the corrections have been applied. The WFS measures

a residual wavefront (ϕ_{res}), after correction by the mirror, according to: $\phi_{res} = \phi_{turb} - \phi_{corr}$.

This architecture has the advantage of simplifying WFS (wavefronts are only measured with small amplitudes) and enabling the WFS to 'see' the DM.

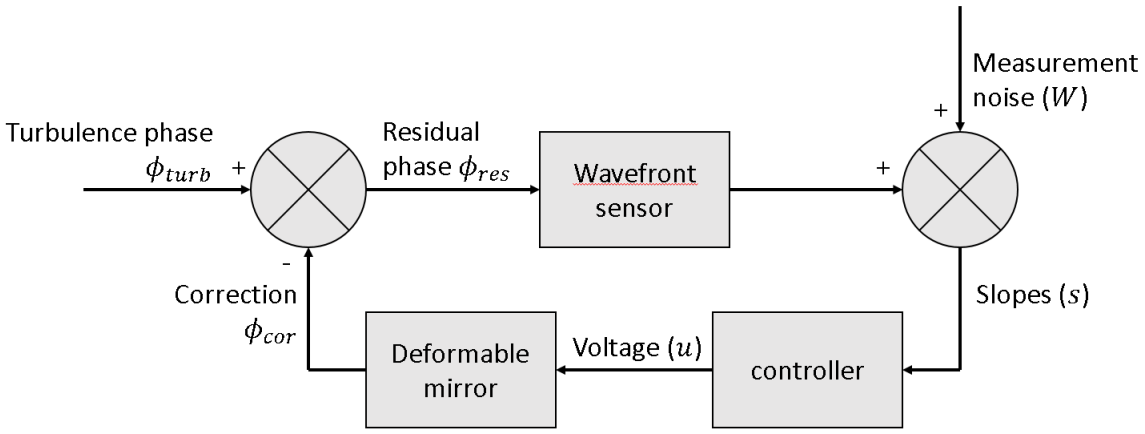


Figure 3.6: Closed-loop adaptive optics system

An alternative architecture to a closed-loop systems is a open-loop configuration. Open-loop refers to systems where the WFS is located before the DM and see the full uncorrected atmospheric wavefront. This is shown in figure 3.7. This configuration can be limited by the dynamic range of the WFS (measuring ϕ_{turb} directly instead of ϕ_{res}).

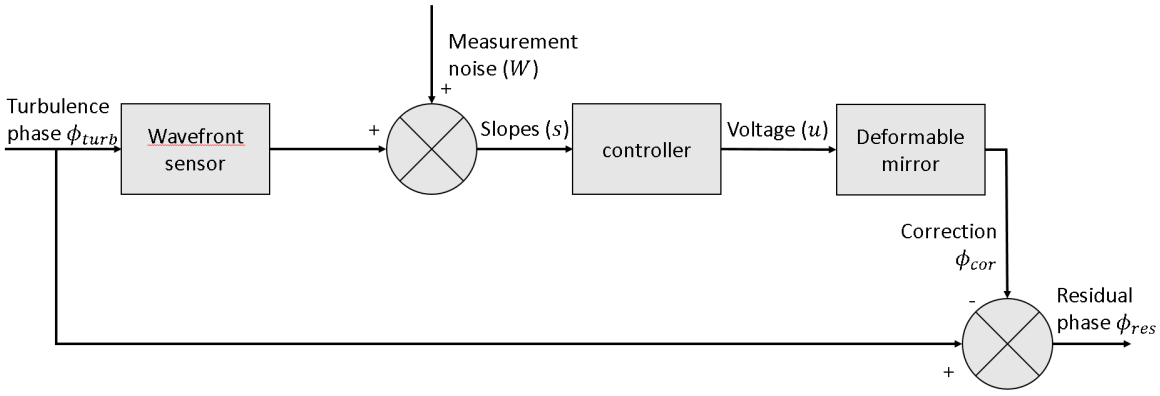


Figure 3.7: Open-loop adaptive optics system.

In open-loop systems, the driven DM shape needs to be accurately

known. In closed loop the WFS images wavefront after the DM and is trying to correct the residual error. In open loop the WFS image uncorrected wavefront and drive the DM into position. If the DM shape that is applied is different from what the AO system is attempted to be apply there will be errors in the system. This can be problematic with certain types of DMs such as one using piezoelectric actuators due to hysteresis[22].

3.7.2 Single conjugate adaptive optics and eXtreme adaptive optics

The performance of single conjugate AO systems (SCAO) is limited by a number of factors including the fitting, temporal and anisoplanatic errors. The latter means that a good correction quality (using a single WFS and a single DM) is limited to a small field-of-view around the guide star (the anisplanatic angle). For this reason, SCAO systems can only be narrow-field.

Extreme Adaptive Optics (XAO) is similar to SCAO but its goal is to achieve extremely good on-axis correction. Typical applications are, for example, direct imaging of exoplanets (combined with a coronagraph). XAO systems not only need to compensate for aberrations inherent to atmospheric turbulence, but also compensate for aberrations created by the instrument itself. Better performance is achieve by reducing the errors of the system, with careful calibration and correction of the instrument related aberrations. It can also be improved by reducing the temporal error ($\sigma^2_{temporal}$) by increasing the loop update frequency and by reducing the fitting error ($\sigma^2_{fitting}$) by increasing the density of actuators on the DM. Instruments like SPHERE[23], which has an

update frequency of 1.2 kHz and uses a DM with 41×41 actuators to achieve Strehl ratio of up to 95%.

3.7.3 Widefield adaptive optics

The goal of widefield AO systems is to increase both the corrected field (limited by anisoplanatism) and sky coverage (limited in SCAO by the NGS magnitude) in order to observe several objects in the field of view simultaneously. There are several different possible configurations for widefield AO: Ground Layer AO, Laser Tomography AO, Multi-Object AO and Multi-Conjugate AO.

3.7.3.1 Ground layer adaptive optics

As stated in section 2.2, the ground layer, being the strongest layer, contributes the most to wavefront distortions. Ground layer adaptive optics (GLAO) is a technique that is designed to only correct for distortions created by the turbulent layer which is the closest to the ground. The DM is generally optically conjugated to the pupil of the telescope, or at a relatively low altitude above it. An illustration of GLAO is presented in figure 3.8. In this particular case, the turbulence is analysed in 3 field directions and uses 1 DM conjugated in the pupil.

Its objective is to ensure a partial (i.e. far from the theoretical maximal resolution of the telescope) but uniform correction in a relatively large field, typically of the order of 2' to 5'. The guide stars can either be natural guide stars, artificial laser guide stars or a combination of both.

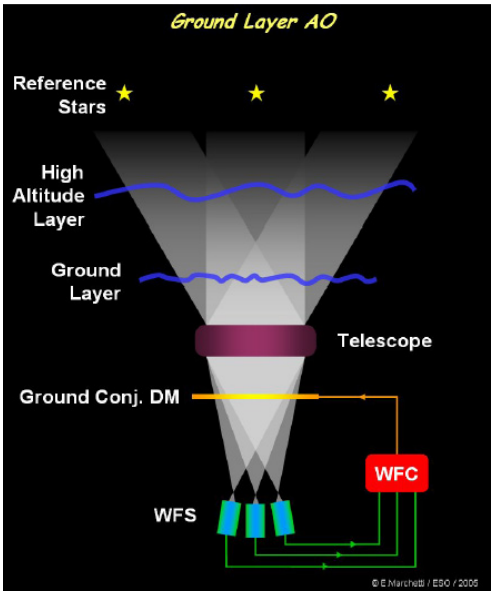


Figure 3.8: Ground layer adaptive optics. Image copyright ESO

3.7.3.2 Laser tomography adaptive optics

Contrary to Multi-Conjugate AO or GLAO, Laser Tomography AO (LTAO) realises its correction in a small field of view, similar in size to that of an SCAO system. A multi-directional WFS analysis is performed to achieve a tomographic reconstruction of the turbulence cylinder in the direction of interest. The correction is then applied using a single DM, generally conjugated to the pupil of the telescope and in a specific direction field. An illustration of LTAO is shown in figure 3.9.

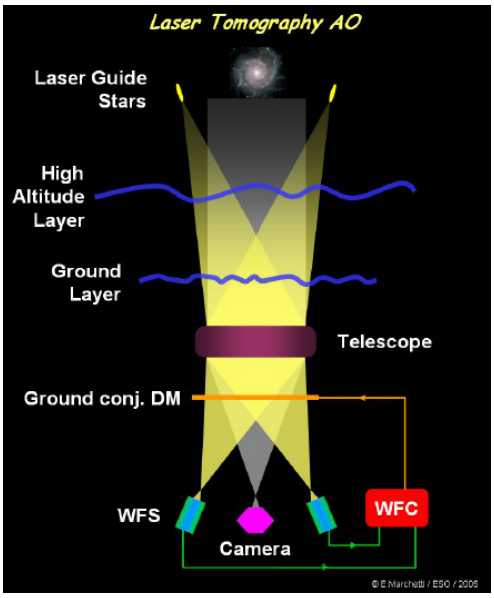


Figure 3.9: Laser Tomography used to obtain on-axis performance using WFS measurements from LGSs located at a distance from the science target. Image copyright ESO

In the case presented here, the turbulence is analysed in 2 directions of the field using laser guide stars. A single DM conjugated to the ground enables the correction in a specific field direction (here on-axis) that is different from the WFS analysis directions.

3.7.3.3 Multi-Object adaptive optics

In astronomy, it is sometimes required to have high levels of correction across very wide fields of view but only for specific locations within that field. Typical objectives of Multi-Object AO (MOAO) is to observe multiple galaxies simultaneously. These galaxies are generally not bright enough to perform wavefront sensing. MOAO is similar to how LTAO performs a tomographic reconstruction of the atmospheric turbulence. Separate DMs are then used to ensure the correction for each of the directions of interest. One of the difficulties of such a system is that the DMs need to operate in open-loop: the WFSs do not see the applied correction. A diagram of its principles is shown in figure 3.10.

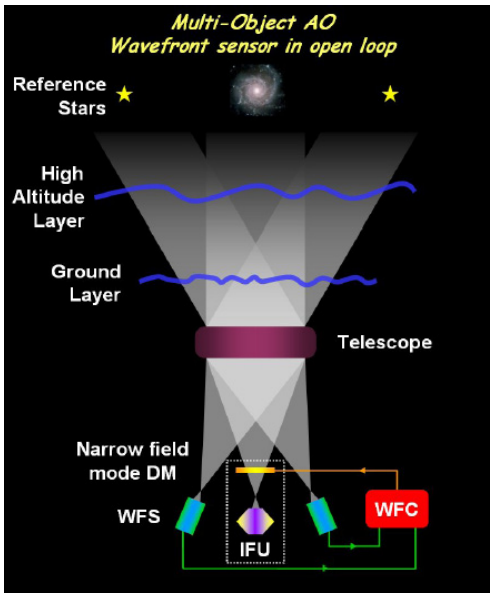


Figure 3.10: Principle of MOAO using 2 WFSs observing NGS, and 1 on-axis DM conjugate to the pupil plane. Image copyright ESO

3.7.3.4 Multi-Conjugate adaptive optics

In Multi-Conjugate AO (MCAO), multiple DMs are used that are conjugated at different altitudes, each compensating for a different turbulent layer (see anisoplanatism). Multiple WFSs are used, combining multiple natural guide stars and laser guide stars in order to have a knowledge of the turbulent volume and its distribution (tomographic reconstruction). The objective of MCAO is to obtain a correction quality close to the diffraction limit for a field of the order of 1' to 2'. The principle of MCAO is presented in figure 3.11.

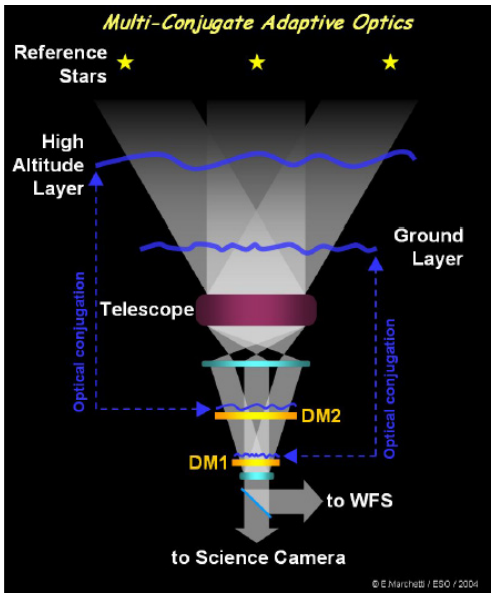


Figure 3.11: Principle of MCAO using 2 DMs (each optically conjugate to a turbulence layer) and 3 WFSs. Image copyright ESO

3.8 Future telescopes - ELTs

One of the current largest optical ground based telescope is the Gran Telescopio Canarias located on the Canary Islands. It has a primary mirror measuring 10.4 m in diameter. This is one of the many 8 to 10 metre telescopes that are in operation. The majority of these telescopes came into operation in the late 1990s or early 2000s. The next series of planned ground based telescopes are referred to as the ELTs. Currently three major observatories are proposed: the Thirty Meter Telescope (TMT), the Giant Magellan Telescope (GMT) and the European Extremely Large Telescope (E-ELT). All these telescopes will have a primary mirror comprising of between 24 and 40 m in diameter.

A selection of the current and planned optical ground based telescopes are shown in figure 3.12. The extreme increase in telescope size with the planned ELTs gives an idea of how these observatories are pushing technology developments both for construction and operations.

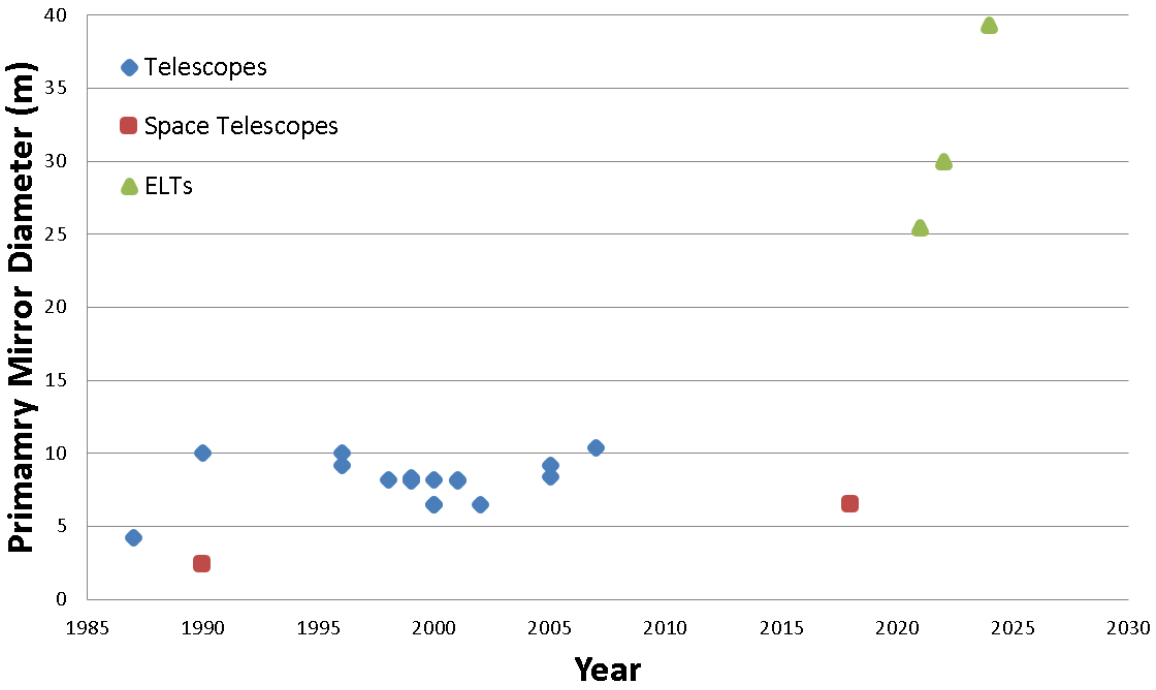


Figure 3.12: Comparison of modern ground and space optical telescopes. Blue shows the current telescopes, green the planned and red space telescopes.

Moving from the current telescope sizes to the ELTs poses numerous challenges regarding the implementation of AO systems. The wavefront reconstruction and the control of systems with large degrees of freedom are some of the most important challenges. This is increasingly relevant as the control frequencies have increased from approximately 100 Hz for the first systems to typically 500-1000 Hz (and even up to a few thousand hertz). Furthermore, new concepts that build on the traditional narrow field single conjugate AO, to create large corrected fields of view are even more complex to implement.

3.9 Conclusion

The aim of an AO system is to reduce the effect of the atmospheric turbulence in real-time on images captured by ground based telescopes. We have introduced the major concepts of an AO system and briefly

mentioned its limitations. We have focused on the temporal and fitting errors, two of many sources of error in AO systems. As telescopes get bigger these two errors are the driving force behind the complexity of AO systems. To reduce the fitting error more actuators are used, which increases the complexity in the AO RTC. The faster you can apply the correction the more the temporal error can be minimised. Combining these effects, future systems will have more actuators and higher loop frequencies, and will require more computing power than ever before.

The focus of the research presented in this thesis is the real-time control for AO (an effort to minimise the temporal error). In this chapter, we have briefly introduced the concept of AO real-time control. In the next chapter (chapter 4), we expand into the area of real-time computing. This will then lead onto an in-depth investigation into the AO RTC in chapter 5.

References

[1] Horace W Babcock. The possibility of compensating astronomical seeing. *Publications of the Astronomical Society of the Pacific*, pages 229–236, 1953.

[2] David Fried. History of adaptive optics, 2013. CFAO summer school.

[3] Corinne Boyer, Vincent Michau, and Gerard Rousset. Adaptive optics: interaction matrix measurements and real time control algorithms for the COME-ON project. In *The Hague'90, 12-16 April*, pages 63–81. International Society for Optics and Photonics, 1990.

[4] F Roddier. *Adaptive optics in astronomy.* Cambridge University Press, 1999.

[5] Jean-Luc Beuzit, Norbert N Hubin, Eric Gendron, Laurent Demailly, Pierre Gigan, Francois Lacombe, Frederic Chazallet, Didier Rabaud, and Gerard Rousset. ADONIS: a user-friendly adaptive optics system for the ESO 3.6-m telescope. In *1994 Symposium on Astronomical Telescopes & Instrumentation for the 21st Century*, pages 955–961. International Society for Optics and Photonics, 1994.

[6] Ben C Platt and Roland Shack. History and principles of shack-hartmann wavefront sensing. *Journal of Refractive Surgery*, 17(5):S573–S577, 2001.

[7] Daniel R Neal, James Copland, and David A Neal. Shack-Hartmann wavefront sensor precision and accuracy. In *International Symposium on Optical Science and Technology*, pages 148–160. International Society for Optics and Photonics, 2002.

[8] Roberto Ragazzoni. Pupil plane wavefront sensing with an oscillating prism. *Journal of modern optics*, 43(2):289–293, 1996.

[9] E Gaviola. On the quantitative use of the foucault knife-edge test. *JOSA*, 26(4):163–163, 1936.

[10] Christophe Vérinaud. On the nature of the measurements provided by a pyramid wave-front sensor. *Optics Communications*, 233(1):27–38, 2004.

[11] Roberto Ragazzoni and J Farinato. Sensitivity of a pyramidic wave

front sensor in closed loop adaptive optics. *Astronomy and Astrophysics*, 350:L23–L26, 1999.

[12] Bruce Macintosh, Mitchell Troy, Rene Doyon, James Graham, Kevin Baker, Brian Bauman, Christian Marois, David Palmer, Donald Phillion, Lisa Poyneer, et al. Extreme adaptive optics for the Thirty Meter Telescope. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 62720N–62720N. International Society for Optics and Photonics, 2006.

[13] James A Georges, Proteep Mallik, Thomas Stalcup, James Roger P Angel, and Roland J Sarlot. Design and testing of a dynamic refocus system for Rayleigh laser beacons. In *Astronomical Telescopes and Instrumentation*, pages 473–483. International Society for Optics and Photonics, 2003.

[14] DM Simonich, BR Clemesha, and VWJH Kirchhoff. The mesospheric sodium layer at 23 s: Nocturnal and seasonal variations. *Journal of Geophysical Research: Space Physics*, 84(A4):1543–1550, 1979.

[15] JR Morris. Efficient excitation of a mesospheric sodium laser guide star by intermediate-duration pulses. *JOSA A*, 11(2):832–845, 1994.

[16] F Rigaut and E Gendron. Laser guide star in adaptive optics-the tilt determination problem. *Astronomy and Astrophysics*, 261:677–684, 1992.

[17] M Tallon and R Foy. Adaptive telescope with laser probe-isoplanatism and cone effect. *Astronomy and Astrophysics*, 235:549–557, 1990.

[18] Olivier Lardière, Rodolphe Conan, Colin Bradley, Glen Herriot, and Kate Jackson. Laser-guide-star wavefront sensing for TMT: experimental results of the matched filtering. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 70154T–70154T. International Society for Optics and Photonics, 2008.

[19] M Séchaud. *Chapter 4: WF compensation devices*, pages 57–99. Cambridge University Press, 1 edition, 1999.

[20] Joel Kubby. Wavefront correction, 2013. International Summer School on Adaptive Optics.

[21] Thierry Fusco, G Rousset, J-F Sauvage, C Petit, J-L Beuzit, K Dohlen, D Mouillet, J Charton, M Nicolle, M Kasper, et al. High-order adaptive optics requirements for direct detection of extrasolar planets: Application to the sphere instrument. *Optics Express*, 14(17):7515–7534, 2006.

[22] Alfredo Dubra, John Massa, and Carl Paterson. Preisach classical and nonlinear modeling of hysteresis in piezoceramic deformable mirrors. *Optics Express*, 13(22):9062–9070, 2005.

[23] T Fusco, J-F Sauvage, C Petit, A Costille, K Dohlen, D Mouillet, J-L Beuzit, M Kasper, M Suarez, C Soenke, et al. Final performance and lesson-learned of SAXO, the VLT-SPHERE extreme AO: from early design to on-sky results. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 91481U–91481U. International Society for Optics and Photonics, 2014.

Chapter 4

Real-time computing

Contents

In this chapter we introduce the main concepts of real-time computing. Real-time computing is a term used to describe systems that are guaranteed to provide a response within a specified time. The actual time constraints will depend on the nature of the system it is trying to control. For adaptive optics (AO), the atmospheric turbulence imposes timescales typically of the order of the millisecond or less. Real-time processing will typically require both parallel computing and fast responses in a time critical manner. In this chapter, we first describe some important aspects of real-time and parallel computing (4.1 and 4.2), and in particular on programming languages (4.3) and hardware (4.4).

4.1 Classification of real-time computing

Real-time systems have to guarantee a response within a specific (generally tight) time frame. It is commonly assumed that this means that it has to be a high performance system as well. Although real-time systems tend to be high performance, this is not always the case. Real-time systems have to work in a very predictable way, and deliver within a specified time frame.

As a very simple example, let us consider a chess program. If given unlimited time, it could calculate all possible moves and choose the best one[1]. If the same computer entered a chess tournament, it would simply be disqualified for running out of time. This means that the program actually needs to calculate the best move within a specific time. This type of system can be described as a hard real-time system, as if the

[1]This actually would be impossible, as the number of possible move combinations in a game of chess is larger than the number of electrons in the observable universe.

computer fails to output its move in the specified time limit, it will be disqualified. While a non tournament chess program only has to provide its move to the user in short (but non specific) amount of time and can be thought of as a soft real-time system. There are three main types of real-time computing on a sliding scale of requirements: hard, firm and soft.

4.1.1 Hard real-time

Hard real-time refers to tasks or systems that are essential to the control loop with a very strict latency requirement. Missing a deadline is a total system failure. An example of a hard real-time system could be the flight management system on a fighter jet called 'fly-by-wire'. To make a fighter jet as agile as possible, they are unstable in one or two axes[14]. A control computer needs to update the controls within a certain time frame to keep the jet in the sky. If this system fails, it could result in the jet crashing. In AO, they are typically separated into two categories: fast and slow hard real-time tasks. A typical example of fast hard real-time is the update frequency of the deformable mirror command vector.

4.1.2 Firm real-time

Firm real-time refers to functions with precise latency requirements where infrequent deadline misses are tolerable but will degrade performance. If too many deadlines are missed, the system will fail.

4.1.3 Soft real-time

Soft real-time is related to tasks with relaxed latency requirements, where missing a deadline will degrade performance in a non-critical way. They are typically background functions running during real-time operations in order, for example, to monitor the observing conditions and to update the control parameters. For example, updating the Fried parameter (r_0) or the average wind speed $\langle v_w \rangle$.

Real-time systems used in AO generally comprise of a combination of soft, firm and hard real-time tasks. Whereas the main AO control loops (i.e. computing the DM commands) are hard real-time task, others are auxiliary and run in parallel. These high-level functions usually require the processing of significant amounts of telemetry data from the real-time control system (RTC) control loops at a time scale much larger than the AO update frequency. Typical examples are the optimisation of loop parameters, the estimation of the system parameters or the computation of offloads.

4.2 Computing power and parallel computing

4.2.1 Moore's law

The advancements in technology have been staggering. Since the invention of the transistor computer in the 1950s, computers chips clock frequencies have gone from 50 MHz up to 5 GHz. In 1965, the co-founder of Intel, Gordon Moore released a paper entitled 'Cramming more components onto integrated circuits'[20]. In this paper, he commented on the future of integrated circuit boards.

The complexity for minimum component cost has increased

> *at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years.* Gordon E. Moore

This notion of the number of components on an integrated circuit board doubling every two years has been labelled Moore's Law. The accuracy of this idea was surprising[17]. Moore's Law, first proposed in 1965 over half a century ago, suggested this exponential growth of components would only continue for ten years. Fifty years after its suggestion, Moore's law is still valid. However, there has been suggestions this is likely to change in the near future. Figure 4.1 presents the technological advancements in microprocessors over the past 40 years. The data (not just figure 4.1.) shows the steady increase of transistor counts (following Moore's law) and the slight increase in single-thread performance. It also shows the clock frequency and typical power usage have started to plateau. Finally, it gives an idea of the steady increase of the number of cores since 2005, following a power law[17].

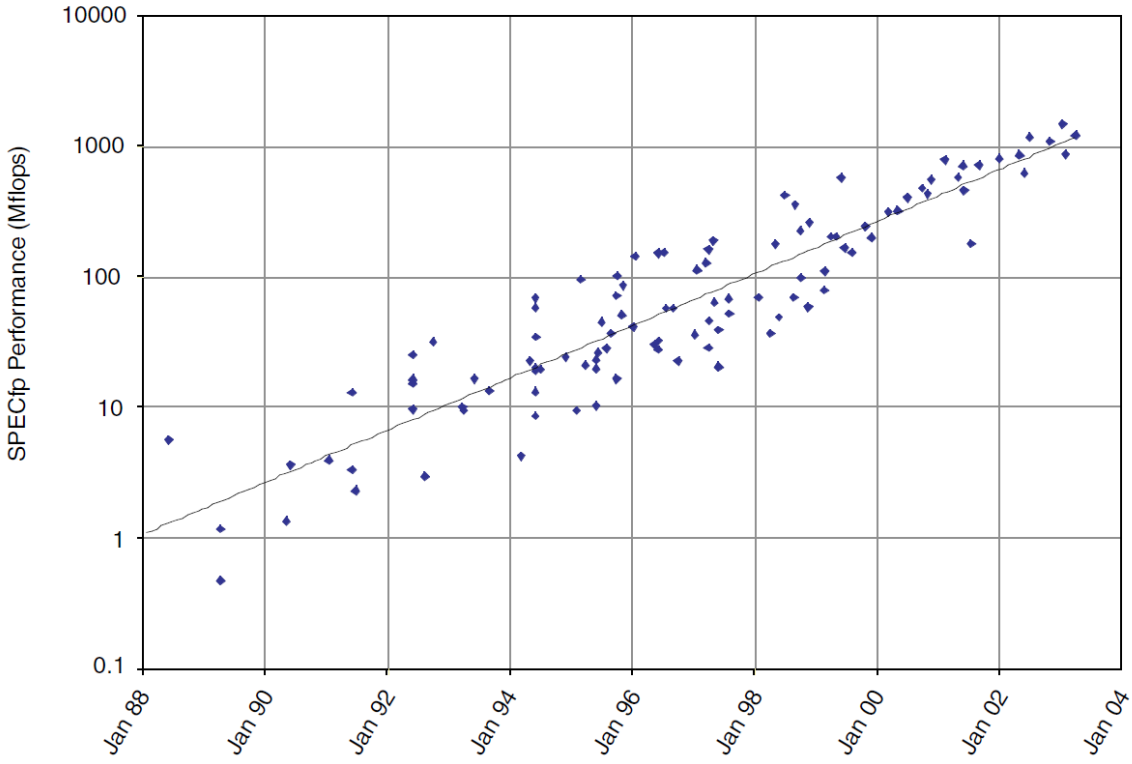


Figure 4.1: Performance of microprocessors over time. Copyright: National Academic Press[4].

4.2.2 Parallel computing and Amdahl's law

Moore's law only focuses on the increased performance of a single chip. Modern computer processors are implementing the parallel computing on a single chip. The majority of computers on the market no longer have a single processing core; they generally have two, four, eight and in some cases even more.

If the task at hand is able to be parallelised, then the processing time can be dramatically decreased. Amdahl's law, first put forward by Gene Amdahl in 1967, states that the maximum speedup that can be achieved from applying more processors and can be calculated by equation (4.1)[1], s is the amount of time spent performing serial tasks, p is the amount of time spent performing parallel tasks (total time

$T = s + p$) and N is the number of processors.

$$\text{speedup} = \frac{1}{s + \frac{p}{N}} \tag{4.1}$$

If we allow the number of processors (N) to go to infinity,

$$\text{speedup}_{max} = \lim_{N \to \infty} \frac{1}{s + \frac{p}{N}} = \frac{1}{s} \tag{4.2}$$

This gives a very pessimistic view of the possible speedup from using large parallel computers. If we have a system with 5% of serial code, then the maximum speedup achievable from a system with an infinite number of processors is $\times 20$. Figure 4.2 shows Amdahl's law for various values of serial code. We see from this that the system approaches an asymptote.

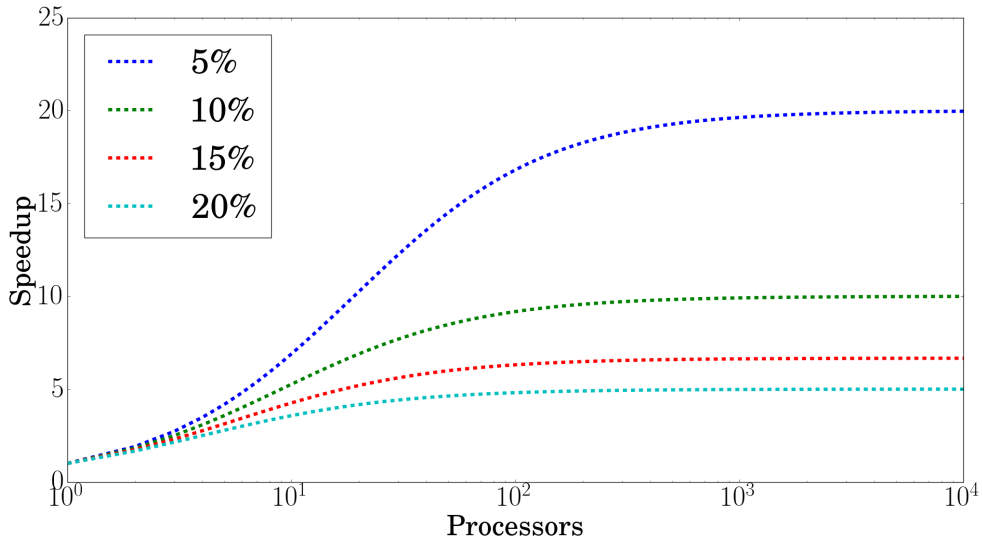


Figure 4.2: Speedup as a function of the number of processors according to Amdahl's law. The different curves represent different percentage of serial code relative to the overall code.

Amdahl's law has been shown to be correct, thought it does address a simple case. It makes the assumption that with the increase in computing resources, the problem size remains constant, which is generally

not the case. As computing resources are added, this generally allows the problem size to scale similarly. When the problem size is increased, the portion of parallel code is also increased (i.e. it increases faster than the serial code sections). This is known as Gustafson's law.

If we let $\beta = \frac{s}{s+p}$, then Gustafson's law can be expressed as equation (4.3)[10].

$$\text{speedup} = N(1 - \beta) - \beta \tag{4.3}$$

This view of parallel computing is much more optimistic than Amdahl's law predicts. The speedup forecast by Gustafson's law is a linear equation ($y = mx + c$), where the gradient $m = 1 - \beta = 1 - \frac{s}{s+p}$. Contrary to Amdahl's law, it has no upper limit for the speedup, and adding computing resources such as more computer processors becomes much more appealing. This is shown in figure 4.3.

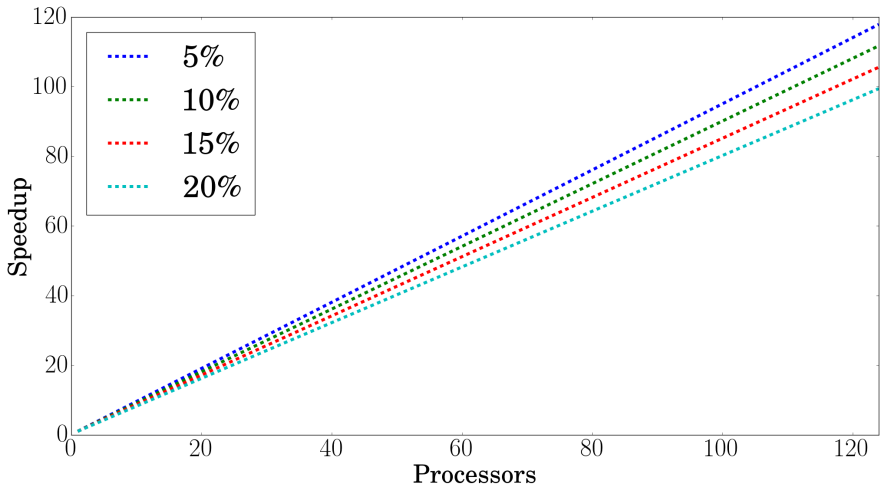


Figure 4.3: Speedup as a function of the number of processors, according to Gustafson's law. The different curves represent different percentages of serial code, relative to the overall code.

4.2.3 Measures of performance and complexity

4.2.3.1 Instruction based performance and complexity metrics

FLOP

To measure and compare the complexity of a given algorithm, the number of floating point operations (FLOP) are used. This can be useful when comparing the complexity of different algorithms performing the same tasks. A floating point operation is a arithmetic operation such as addition, subtraction and multiplication performed on floating point variables. Integer and floating point operations are performed differently in the CPU. Other operations such as division, powers (i.e. x^n) and roots (i.e. $\sqrt[n]{x}$) all are more complicated in the computer, and require more than one FLOP to be computed. For example, divisions calculated by computers typically are assigned the value of 4 FLOPs, and powers (i.e. x^n) require n FLOPs.

FLOPs are a good metric to compare the idealised complexity of different algorithms. When comparing the performance of these on a real processor, or comparing the same algorithm running on two different processors, FLOPs per second are used (FLOPS). A theoretical value for the maximum achievable FLOPS for a given hardware can be calculated using equation 4.4. $\frac{\text{FLOPs}}{cycle}$ is the number of floating point operations achievable by a single processing core in a single clock cycle.

$$FLOPS = \frac{\text{FLOPs}}{cycle} \times \text{clock frequency} \times \frac{\#\text{ cores}}{socket} \times \#\text{ sockets} \quad (4.4)$$

The achievable FLOPS can then either be used to compare the performance of a processor with another, or to compare the actual performance of an algorithm with its theoretical maximum.

MACS

An alternative to FLOPs and FLOPS is the Multiply-accumulate (MAC) per second (MACS). This can be used in the same way FLOPS can, for measuring complexity or performance. The MAC unlike a FLOP is a specific operation. This operation is a multiple and an accumulate, as shown in equation (4.5).

$$a = b + c \times d \tag{4.5}$$

. This can be a useful metric, as in a modern processor this type of operation can be performed in a single clock cycle.

Operational Intensity

Operational Intensity (I) is another useful metric and is defined in equation (4.6).

$$I = \frac{W}{M} \tag{4.6}$$

It is the ration of work (W) to memory traffic (M). Work is defined as the number of numerical operations performed per second. This can include integer operation and floating point operation. Though in many cases it is defined as the number of FLOPS. M is the amount of memory that is required for the calculation.

This metric is useful as it conveys the notion of memory bandwidth. In section 4.2.3.2 we discuss how computational power is increasing faster than memory bandwidth. This suggests the limiting factor of some calculations maybe the ability to transfer data from memory to the CPU rather than the raw computing power. This metric is able to hold information about both the computing power required as well as

the memory bandwidth required.

4.2.3.2 Memory bandwidth

The maximum theoretical stated FLOPS value for a processor is rarely achievable. This is due to the stated value only assuming the maximum number of instructions that can be performed. Many algorithms are limited, not just by the number of calculations achievable, but also by the ability of the computer system to be able to transfer the required memory values to the processor in time. If the processor has to wait for memory to be retrieved during calculation, then we refer to the system as being memory bandwidth limited.

Over time, both the CPU frequency and DRAM speeds have increased. The majority of the increase has been in the CPU frequency rather than in the DRAM. This has led to more systems becoming memory bandwidth limited. Figure 4.4 shows how the CPU frequency has increased faster than the increase in DRAM speeds.

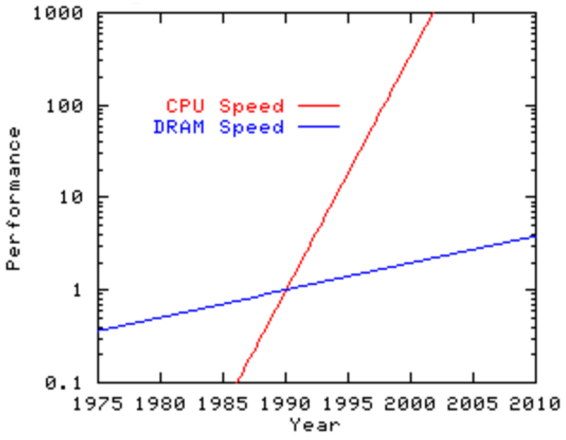


Figure 4.4: A comparison of the increase in CPU frequency and DRAM speed. Copyright: John McCalpin (University of Virginia)

STREAM

The STREAM Benchmark is a simple benchmarking program to calculate the sustainable memory bandwidth and corresponding performance of simple vector calculations[16]. It uses four different calculations, and measures the processing times to infer the memory bandwidth of the processing unit. The four calculations being used are shown in table 4.1.

Table 4.1: The STREAM calculations

name	Kernel	bytes/iter	FLOPS/iter
COPY	$a_i = b_i$	16	0
SCALE	$a_i = q * b_i$	16	1
SUM	$a_i = b_i + c_i$	24	1
TRIAD	$a_i = b_i + q * c_i$	24	2

The STREAM benchmark measures the memory bandwidth from the main memory and not cache memory. To measure the main memory bandwidth, the array size used must be at least four times that of the total cache memory of the chip.

The STREAM test can only give bandwidth for code that is not specifically tuned for specific hardware architectures. This enables code to be compared on many different processing chip types. Optimising the code for specific hardware can obviously lead to higher memory bandwidths. STREAM can also be used on single, multi and many-core processing architectures with the use of the OpenMP (see section 4.3.1.2).

4.3 Programming languages

Many people regard Ada Lovelace as the first computer programmer. In 1843 Lovelace published a document describing the design of Charles

Babbage's Analytical Engine. Within her notes, was what is known as the first computer program[13], a series of commands for the Analytical Engine to compute the series known as Bernoulli numbers. Unfortunately, the Analytical Engine was never built during her life time due to Babbage being unable to raise the required funds.

A computer language is a method to communicate instructions to a computer. These instructions are typically converted from the format in which the developer writes them, to a form understood by the computer called machine language. Machine language is the lowest level instruction used by the computers central processing unit (CPU), and this is in the form of binary or hexadecimal instructions. Each instruction in machine code refers to some very simple and specific tasks. These tasks vary from loading memory, to branching sequences or for the arithmetic logic unit (ALU)[2] to perform a calculation. Typically, computer programmers do not work in machine language as this is too low level and development time is long and difficult. Computers can typically carry out billions of instructions per second, and require many commands to perform even the most trivial tasks. By trivial we mean in the reference to the developer. For example, at a high level taking a drink from a cup is trivial, when broken down to all the individual instructions it becomes a much more complex task requiring many separate individual components. This level of abstraction, making common tasks simple is typically why higher level computer languages are used.

By higher level computer languages, we refer to computer languages that abstract the developer away from these lowest level of instructions and allow them to focus on solving the bigger problems. Computer

[2]A computers CPU is made up for two main units. The ALU, which performs arithmetic calculation and, a control unit which fetches the instructions from memory and carries them out.

programming languages give an intermediate step where the developer writes their program, which is then 'compiled' into machine code. This compiling step takes the computer code written by the developer, and translates it into the lower level machine code that is executed by the computers CPU.

Computer programming languages have advantages and disadvantages, as they have been developed for specific usages. C/C++ is a high level procedural language, but gives the programmer access to lower level features such as memory handling. C was developed in the 1970s at Bell labs[22] and C++ was developed to give a C like language with use of Object Orientation design[3].

Unlike Java, C/C++ is compiled down to machine code by the developer before it is run. Languages such as C/C++ are typically preferred for time critical tasks over those such as Java. This is due to not having 'Just-In-Time' compilation, as well as not using Java's automatic memory handling, and uses a garbage collector to clean up memory no longer being used. This garbage collector decides when it needs to clean up the memory, and can cause unpredictability in the performance. This means that the C/C++ developer has much more control over critical parts of the code.

4.3.1 Parallel API

Historically, computing hardware has only had a single CPU or processing core. This has meant that computer programming languages were developed and designed to work with only a single processing thread. It was not until the early 2000s that the first multi-core processors[4]

[3]C++ is a superset of C. This means that any valid C program is also a valid C++ program.

[4]processor with multiple CPUs on the single chip.

were beginning to become available, although other forms of parallel computing existed previously. In 2001, IBM released the power 4[23]. It was not until 2005, when AMD released the Athlon 64 X2 and Intel release the Pentium D[8], that multiple processing cores started to became common place.

With the introduction of multiple cores, computer programs are now able to simultaneously perform more than a single task. To make this available to older languages, many extensions and APIs have been developed.

- pthreads
- OpenMP
- OpenCL
- OpenACC
- Vectorisation

4.3.1.1 pthreads

Pthreads is a standard threading library for UNIX systems in the C language, for shared memory multiprocessor CPU architecture. Pthreads (POSIX threads) is compliant with IEEE POSIX 1003.1C standard. It allows multithreaded applications to be build within standard C, and offers the tools to create, synchronise and destroy threads.

Pthreads have access to the shared memory of the system, and pointers to memory can be passed between threads. As opposed to OpenMP (section 4.3.1.2), which operates a fork-join model, Pthreads offers the ability to create and destroy threads, detach them and synchronise them. This gives the developer complete control over how the thread

architecture is developed. With this large degree of control, longer development times in comparison to OpenMP are likely. The performance gains make it more appropriate for use in time critical scenarios.

Although Pthreads is specified in an IEEE standard, the implementations on different systems can vary and may operate differently on different hardware. This can lead to slightly different behaviour, or in extreme cases may fail or produce incorrect results. This can be as simple as a limit on the maximum number of threads allowed on a system, which is less than the number of threads specified in the code, causing the program to fail. In C++ (C++11 onwards) the library 'Threads' offer similar functionality as the Pthreads library.

4.3.1.2 OpenMP

Open Multi-Processing (OpenMP) is a cross-platform API for developing shared memory multiprocessing applications. OpenMP adds a selection of keywords accessed through '`pragma`s' that enable the developer to quickly develop applications that take advantage of the multiple cores available on modern CPUs. Listing 4.1 shows a very simple parallel for loop that will be split between the number of CPUs available on the computer.

Listing 4.1: OpenMP code example.

```
int a[100000];
#pragma omp parallel for
for (int i = 0; i < 100000; i++) {
  a[i] = 2 * i;
  }
```

This programming is a fork-join model, which means that for these

sections, new threads are created to do the specified work, then they are destroyed at the end of each section. In this model the processing forks and rejoins multiple times through the code. An example of this can be seen in figure 4.5[5].

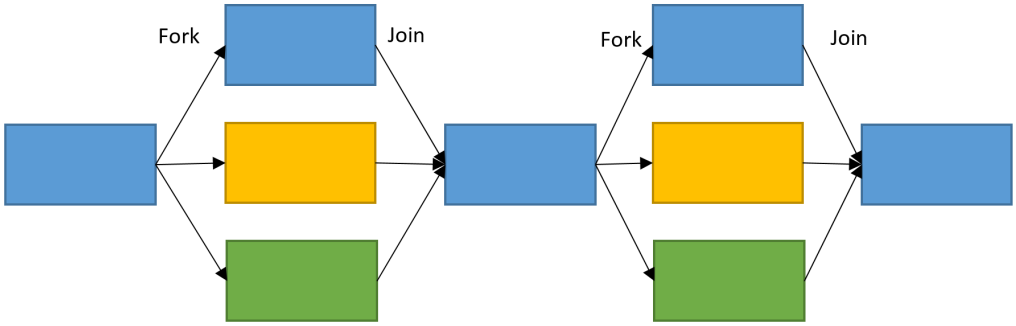


Figure 4.5: An example of fork-join processing produced by OpenMP.

As well as OpenMP or Pthreads, there are other similar libraries provided by companies such as Intel. For example, Intel Threading Building Block (Intel TBB) gives similar functionality to OpenMP, with '`parallel_for`' or '`parallel_reduce`'. This is implemented as a library rather than a language extension. This fork-join model is also used in Intel's Cilk Plus parallel API.

4.3.1.3 OpenCL

Open Computing Language (OpenCL) a cross-platform framework that allows the execution on many different computational hardware such as CPUs, GPUs, Xeon Phis, DSPs and FPGAs. It is aimed at allowing hardware independent development. The developed code can then be compiled and executed on any hardware available, with minimum tweaking. It is also aimed at allowing parallel processing so is able to target many- and multi-core architectures as well has more exotic hardware of GPUs and FPGAs.

Many people have studied the efficiency of OpenCL against performing the same function using other tools such as CUDA. In a study by Karimi[12], it was found that performing the same code was faster for both transfer of data and processing of data on a GPU using CUDA over OpenCL. It concluded that for high performance or time critical tasks CUDA, is the better choice. For non high performance or time-critical applications the performance difference is not enough to force users to use CUDA and the choice should be made based on the developer's knowledge and familiarity with both tools.

4.3.1.4 OpenACC

Open accelerators (OpenACC) is an API that is designed to simplify the development of parallel programming on heterogeneous hardware. This is similar to how OpenMP works as well as the compiler assisted offload mode of the Xeon Phi. The developer uses keywords to mark regions to be offloaded to hardware accelerators to allow that region of code to be performed by the high performance parallel hardware. While OpenMP only supports parallelisation across a single architecture, OpenACC supports the use of hardware accelerators and heterogeneous computing platforms.

OpenACC provides a very interesting new tool to help develop parallel code, and will simplify the development of these hardware accelerators. This current generation of Xeon Phis (Knights Corner)[5] is in direct competition with GPUs. GPUs require development in NVIDIA CUDA. A tool such as OpenACC is invaluable for testing the performance of both these technologies, while only developing code once and interchanging the hardware.

[5]At time of writing, though Knights Landing is now released

AO RTCs typically have the problem that they run on outdated hardware. This is because of the long development time and long working life time of the instruments. The computational hardware is chosen early in the development cycle, so by the time the instrument is on-sky the hardware is out dated. Spares of all computational hardware are bought in case of hardware failure. OpenACC may be beneficial to systems being developed on one hardware but if the hardware breaks new COTS hardware can be purchased and will work even if not identical to the original. The performance however would still need to be reevaluated.

4.3.1.5 Vectorisation

Vectorisation is a case of parallelisation where a single command can be propagated to multiple sets of data. This can be automatic in the case of Intel compilers and processors, or it can be explicit in the case of AltiVec on powerPCs.

With vectorisation, a single command such as addition, subtraction or multiplication can be performed on an entire data set. For example, in listing 4.2, an addition between $vector1$ and $vector2$ is performed and automatically translated to make use of the vector units on the Intel based processor. Auto vectorisation allows the code to be performed by the SIMD registers. These registers are 128 bits wide or four single precision floating point variables. This can lead to a four time speedup over that of the same computations being computed sequentially.

Listing 4.2: Auto vectorisation using the SIMD registers.

```
for (i=0; i<n; i++){
  resultVector[i] = vector1[i] + vector2[i];
```

```
}
```

When using explicit vectorisation, the user has to specify that the command be performed on the vector registers. For AltiVec powerPCs, listing 4.3 shows an example of a function designed to perform vectorised addition on a powerPC. This leads to more work for the programmer, but ensures that section of code that are specified for vectorisation are actually performed on the vector registers.

Listing 4.3: Example of explicit vectorisation on AltiVec powerPCs.

```
resultVector = vec_add(vector1, vector2).
```

4.3.2 Real-time performance and the operating system

The biggest difference between real-time computing and high performance computing is the need for predictability. Frequently, real-time also refers to high performance, as the result may need to be calculated as quickly as possible to meet the required deadlines. High performance does not always mean real-time, as the goal can be to process as much as possible, as quickly as possible and the deterministic response time may not be important.

There are many factors that come into play when developing real-time applications, not just in the code being developed but also on the platform it is being deployed on. The operating system (OS) is likely to be the biggest factor in the deterministic nature of how the application performs.

4.3.3 Operating systems and the scheduler

The operating system in a computer can be seen as the 'brain', and it performs the background functions that are not specified by the developer. The operating system performs the basic tasks such as scheduling and controlling peripherals.

One of the main jobs of the operating system, is to schedule tasks and decide where these tasks need to be executed. Some of these tasks are specified by developers such as running code, others are background tasks that are needed to keep the computer running. These tasks are held in a job queue. This means there are generally more tasks than there are cores. To get around this limitation, the OS uses a scheduler to decide which of the tasks in the job queue to perform next and where (e.g. which core).

This can lead to the interleaving execution and interruption of tasks on a core. This interruption of a task to begin the execution of a new job is called context switching. When this occurs the current state of the task being run is put in memory and the next task's 'context' is loaded from memory and can begin. Context switching generally does not cause much of a problem as lots of small tasks are being loaded in and out of memory.

To decide what to run next, there are many different schemes the scheduler can use. Here is a list of the most common scheduling schemes:

- First come first serve
- Shortest job next
- Priority scheduling
- Shortest time remaining

- Round robin scheduling

First come first serve First come first serve scheduling is the simplest form of scheduling. Jobs are executed in the order they arrive, which generally gives poor performance and causes high wait times for jobs. A long low priority job may block shorter but more important jobs.

Shortest job next Shortest job next produces good results when all CPU processing times are known for each job. However, in interactive systems where job CPU time is variable (or unknown) this method is impossible to implement.

Priority scheduling Priority scheduling uses the jobs assigned priority to decide which job to run next. Each job is assigned a priority. This can be determined through various means such as CPU time, memory or other requirements. The job with the highest priority is executed next. If two jobs have the same priority, then they are executed in a first come first served scheme. Priority can be assigned dynamically or be static. In static priority systems, once a job has a priority assigned to it, it remains constant. Dynamic refers to priority of processes changing after they have been assigned.

Shortest time remaining Shortest time remaining is a preemptive version of the shortest job next. In this mode, if a long process is being performed and a process with a shorter time to completion arrives, then this new process with the shorter time to completion can interrupt the current process. As with the shortest job next, Shortest time remaining cannot be performed on interactive jobs where CPU time is unknown.

Round robin Round robin scheduling all processes are provided with a fixed time for them to process. After this fixed time, if another job is waiting, the original job is interrupted and another process is executed. The states of each process are saved with context switching.

The scheduler is also in charge of deciding which cores each of the jobs are to be executed on. The developer can influence this by assigning the affinity of the desired process.

4.3.3.1 Real-time operating systems

The goal of a real-time operating system is to run tasks for real-time applications. They can either be a hard real-time system (i.e. delivering results in a deterministic manner) or a soft real-time system (i.e. generally delivering results in the allocated time). Typically, the non-real-time OS schedule to optimise the overall throughput. For real-time systems, the aim is to optimise for predictability.

For real-time systems a version of priority scheduling is used. This is done to force time-critical processes to preempt and interrupt other processes[15]. This will allow the most important tasks to be preformed when they are available, and non-critical tasks are delayed or interrupted.

4.4 Computational Hardware

4.4.1 FPGA

FPGAs are integrated circuits with the capability of being programmed after manufacturing to complete a given task. FPGAs were originally programmed in low level Hardware Description Languages (HDLs) such as Verilog or VHDL. This tends to be difficult as they are programmed

with only a small amount of hardware abstraction. New alternatives, such as OpenCL (see section 4.3.1.3), are now available which add a level of abstraction from the hardware. OpenCL allows programmers to develop portable code in C, and to take advantage of both the performance and power efficiency of FPGAs and of CPUs.

FPGAs have much lower clock frequencies than other computational hardware, such as CPUs or GPUs. FPGAs sit in the range of 250-300 MHz while GPUs tend to be 1200-1400 MHz and CPUs can be 3000-5000 MHz. Even though the GPUs clock rate is higher, the performance of the FPGAs maybe closer to 4.18 faster when preforming certain tasks[7].

FPGAs have the potential to use less power than conventional processors, and thus produce less heat. Due to the structure, some tasks are more easily implemented into FPGAs than others. It has, however, been shown that FPGAs can increase speed by a factor of twenty or more[25]. FPGAs excel at tasks that require a small amount of processing to a large amount of data with strict time constraints, while CPUs are better at performing large amounts of processing on small amounts of data. FPGAs are commonly used in video processing applications, such as TV for adding overlays or for HDMI drivers.

FPGAs have been used in AO RTC systems to perform tasks such as front-end wavefront processing[9] and communication infrastructure[6]. Although very efficient and reliable, the FPGA design doesn't allow the hardware platform to be easily scalable to accommodate the needs of the different projects. In addition, they are difficult to maintain and upgrade to follow technological evolution. Despite the recent availability of programming languages such as OpenCL, the FPGA development

environment makes the development of an efficient and reliable final product relatively difficult to a non-specialist.

4.4.2 CPU

CPUs offer a platform for general purpose computing. They are able to be easily programmed and customised for the desired task. The CPU is the heart of the computer and has to control all background tasks as well as the user defined tasks. This CPU time sharing leads to interruptions in the processing and higher latency than in other forms of processing.

CPUs have previously only been a single high-frequency core, but multi-core processors are becoming common place. The multi-core aspect allows the programs to split jobs into multiple parts to be run on separate tasks from the same memory.

While CPUs tend to have higher performance variability than technologies such as FPGAs or DSPs, they do allow faster development times. Many modern AO RTC systems are using CPUs as they offer fast development times and high performance[3]. Some, on the other hand only use them for the non time-critical tasks, performing time-critical tasks on more deterministic hardware[6].

4.4.3 GPU (with GPU Direct)

Graphical Processing Units (GPUs) entered the commercial world in 1999 with the NVidia GeForce 256, and are dedicated processing units to render graphical interfaces. Since the introduction of GPUs, they have grown from a single chip to cards with over a thousand cores in just over ten years[11].

When GPUs were released some researchers had the idea of using these platforms to perform non graphical computing. This was done by trying to map their models to the operation of the GPU[26]. The success of this lead to the release of the Compute Unified Device Architecture (CUDA). CUDA is a development platform in C/C++ to allow the programming of GPUs, similar to programming on CPU but with access to a highly parallel architecture[18]. This allows high level programming with a large amount of abstraction from the hardware.

A major obstacle with using technologies such as GPUs is data transfer. Unlike CPUs, FPGAs or DSPs, GPUs are not standalone computer. By this we mean GPUs are typically connected to a CPU via the PCIe bus. For the other technologies, data can arrive directly on the processor to be processed. For GPUs, data arrives first on the CPU, which then copies the data across into the GPU memory. Figure 4.6 shows the flow of data required in order to reach the GPU. This shows that the data first arrives into the CPU memory, then is taken out of the memory and sent to the GPU. Once a result has been calculated and is being transferred away from the computer, this processed is performed again in reverse.

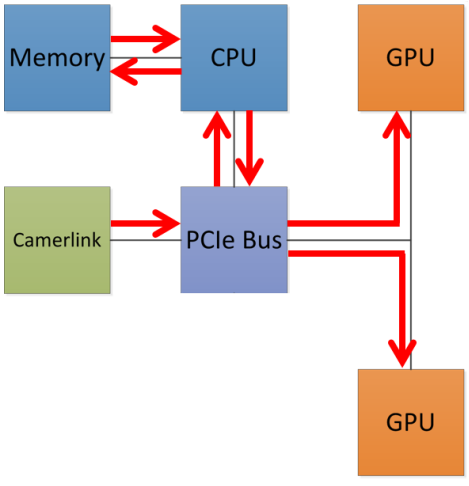


Figure 4.6: A data flow diagram for getting data onto a GPU.

There is a technique that can help mitigate this issue called GPU direct. GPU direct gives the developer the ability to directly access the memory of the GPU under certain circumstances, therefore bypassing the CPU. The flow of data is shown in figure 4.7. Using GPU direct, data can arrive on the GPU much more quickly and reliably in time it is going through the CPU.

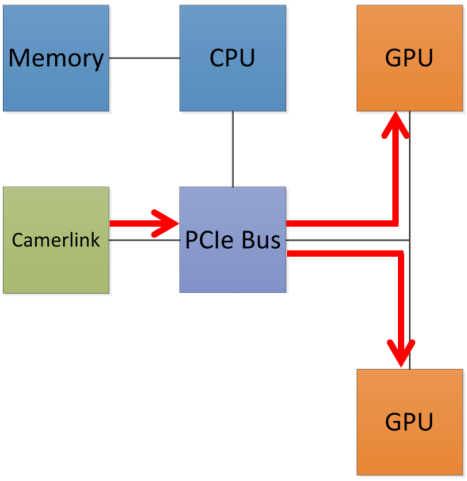


Figure 4.7: A data flow diagram for getting data onto a GPU using GPU direct.

In an AO RTC this is being investigated to see if it mitigates some of the problems associated with the transfer of data into the memory of GPUs. Groups have collaborated with camera manufactures, using data transfer methods such as Camera Link to be compatible with GPU direct[24]. Other groups have developed custom FPGAs cards to allow for GPU direct use[19].

4.4.4 Digital Signal Processors

Digital Signal Processors (DSP) are microprocessors with specialised architectures designed to process digital signals. When processing digital signals, a large number of mathematical operations are preformed quickly on a set of data samples.

Traditionally, DSPs have been used for audio[21, 2] or video processing, where a signal needs low latency processing. The signal arrives on the DSP and is converted from an analog signal to digital signal, using an analog to digital converter (ADC). It can then be processed by the DSP. Once the signal has been processed, it will convert back to analog signal using a digital to analog converter (DAC). This data flow is illustrated in figure 4.8. DSPs are designed to perform tasks in a real-time environment and run a bare minimum OS.

Figure 4.8: A standard DSP flow diagram

DSPs have been used in AO RTC system to performs fast memory intensive tasks[6] such as the matrix-vector multiplication. DSPs have a faster development cycle than FPGAs, but are still slower than CPUs.

4.4.5 Internal Interconnects

All the hardware in section 4.4 have different strengths and weaknesses. In this regard, modern systems tend to be heterogeneous with many different kinds of computational hardware. These computers tend to have hardware acceleration technology that can be used for specific tasks that require more computational power than a traditional CPU can supply. Modern real-time control systems tend to use combinations of GPUs, FPGAs and DSPs to achieve the required performance.

The increase in heterogeneous and distributed computing brings the question of how the data is being transferred around the computing system. Table 4.2 is a comparison of some of the most common internal interconnects found in heterogeneous and distributed computers

systems.

Table 4.2: A comparison of internal interconnects transfer rates

interconnect	transfer rate (MB s^{-1})
PCIe 2.0 (x1, x4, x8, x16)	500, 2000, 4000, 8000
PCIe 3.0 (x1, x4, x8, x16)	984.6, 3938, 7877, 15754
1 GbE	125
10 GbE	1250
InfiniBand (x1,x4)	3125, 12500
Omni-Path	12500
NVIDIA link	20000

Peripheral Component Interconnect Express (PCIe) is the most common interconnect found inside almost all desktop computers. This is the standard type of connection used on co-processors such as GPUs and Xeon Phis. This interconnect has a specific number of lanes that can be used for transferring data, and the more lanes that are used the higher the transfer rate. Typically GPUs and other co-processing cards have x16 lanes to allow for maximum transfer. These typically are short cable lengths in the internals of the computer and provide high data transfer rates.

Omni-Path and InfiniBand fill very similar roles with connecting servers and computers over local networks. This is a role that Ethernet cables can fill as well. Though even the 10 GbE data transfer rate version is much slower than either Omni-Path or InfiniBand. Each of these technologies offer network switches as well as the interconnects to allow large systems.

Two different options are available to GPUs to enable fast interconnects. The first is the use of GPU direct, which has been covered in section 4.4.3. This is a relatively complex solution and requires large

amounts of development time. The second solution is the new NVLink, a point-to-point communication protocol between a GPU and a CPU. This interconnect can be used to transfer data between GPUs, between the CPU and GPUs or any combination. NVLink is a promising new technology advertising a data transfer rate of 20 GB s^{-1}(of 40 GB $s^{-1}$for bi-directional links). This is a proprietary technology designed to work with Nvidia GPUs and may not be a solution to transfer data between servers.

4.5 Conclusion

In this chapter, we have introduced the topic of real-time computing and discussed some of the major building blocks and important topics in real-time computing. Real-time system focus is on temporal determinism as well as the functional behaviour. We have introduced both software and hardware technologies that are being used in modern systems. In the next chapter, we take what has been covered in this chapter and chapter 3 and cover the basics of real-time computing for AO.

References

[1] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967.

[2] Radu Arsinte, Attila Ferencz, and Costin Miron. DSP based system for real time voice synthesis applications development. *arXiv preprint arXiv:0803.0197*, 2008.

[3] AG Basden and RM Myers. The Durham adaptive optics real-time controller: capability and extremely large telescope suitability. *Monthly Notices of the Royal Astronomical Society*, 424(2): 1483–1494, 2012.

[4] National Research Council et al. *Getting up to speed: The future of supercomputing*. National Academies Press, 2005.

[5] Tim Cramer, Dirk Schmidl, Michael Klemm, and Dieter an Mey. OpenMP programming on intel xeon phi coprocessors: An early performance comparison. In *Proceedings of the Many-core Applications Research Community (MARC) Symp. at RWTH Aachen University*, pages 38–44. RWTH Achen University, 2012.

[6] Enrico Fedrigo, Robert Donaldson, Christian Soenke, Richard Myers, Stephen Goodsell, Deli Geng, Chris Saunter, and Nigel Dipper. SPARTA: the ESO standard platform for adaptive optics real time applications. volume 6272, pages 627210–627210–10, 2006. doi: 10.1117/12.671919. URL `http://dx.doi.org/10.1117/12.671919`.

[7] Christopher W Fletcher, Ilia A Lebedev, Narges B Asadi, Daniel R

Burke, and John Wawrzynek. Bridging the GPGPU-FPGA efficiency gap. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, pages 119–122. ACM, 2011.

[8] David Geer. Chip makers turn to multicore processors. *Computer*, 38(5):11–13, 2005.

[9] SJ Goodsell, E Fedrigo, NA Dipper, R Donaldson, D Geng, RM Myers, CD Saunter, and C Soenke. FPGA developments for the SPARTA project. In *Optics & Photonics 2005*, pages 59030G–59030G. International Society for Optics and Photonics, 2005.

[10] John L Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

[11] Daniel R Johnson, Matthew R Johnson, John H Kelm, William Tuohy, Steven S Lumetta, and Sanjay J Patel. Rigel: A 1,024-core single-chip accelerator architecture. *IEEE Micro*, (4):30–41, 2011.

[12] Kamran Karimi, Neil G Dickson, and Firas Hamze. A performance comparison of CUDA and OpenCL. *arXiv preprint arXiv:1005.2581*, 2010.

[13] Eugene Eric Kim and Betty Alexandra Toole. Ada and the first computer. *SCIENTIFIC AMERICAN-AMERICAN EDITION-*, 280:76–81, 1999.

[14] Jaynarayan H Lala and Richard E Harper. Architectural principles for safety-critical real-time applications. *Proceedings of the IEEE*, 82(1):25–40, 1994.

[15] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[16] John D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25, December 1995.

[17] Gordon E Moore et al. Progress in digital integrated electronics. *IEDM Tech. Digest*, 11, 1975.

[18] NVIDIA. Cuda zone, 2016. URL `"https://developer.nvidia.com/cuda-zone"`. Accessed: 2016-07-28.

[19] Denis Perret, Maxime Laine, Damien Gratadour, Arnaud Sevin, and Bertr Le Ruyet. A low-latency link between the nodes of a heterogeneous real-time computing platform for adaptive optics. volume 1, 2015.

[20] I PRESENT. Cramming more components onto integrated circuits. *Readings in computer architecture*, page 56, 2000.

[21] David Reinhardt and Robert C Maher. A real time DSP kernel for concurrent audio tasks. In *Audio Engineering Society Convention 105*. Audio Engineering Society, 1998.

[22] Dennis M Ritchie. The development of the C language. *ACM SIGPLAN Notices*, 28(3):201–208, 1993.

[23] Joel M Tendler, J Steve Dodson, JS Fields, Hung Le, and Balaram Sinharoy. POWER4 system microarchitecture. *IBM Journal of Research and Development*, 46(1):5–25, 2002.

[24] Tuan N. Truong, Antonin H. Bouchez, Rick S. Burruss, Richard G. Dekany, Stephen R. Guiwits, Jennifer E. Roberts, Jean C. Shelton, and Mitchell Troy. Design and implementation of the PALM-3000 real-time control system. volume 8447, pages 84472F–84472F–9, 2012. doi: 10.1117/12.927867. URL `http://dx.doi.org/10.1117/12.927867`.

[25] Wim Vanderbauwhede, Sai Rahul Chalamalasetti, and Martin Margala. Throughput analysis for a high-performance FPGA-accelerated real-time search application. *International Journal of Reconfigurable Computing*, 2012:1, 2012.

[26] Enhua Wu and Youquan Liu. Emerging technology about GPGPU. In *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*, pages 618–622. IEEE, 2008.

Chapter 5

Real-time computing in adaptive optics

Contents

In this chapter, we present specific aspects of real-time computing in the context of adaptive optics (AO). We first give a general overview of the real-time controller (RTC) architecture (section 5.1), splitting it into two main modules. The first module is the wavefront processing unit (WPU) (section 5.2), which covers the wavefront sensor (WFS) data reduction and determining the wavefront slopes. The second is the wavefront reconstruction module (section 5.3) which computes the deformable mirror (DM) commands from the wavefront slopes. We give examples of RTC architectures both for systems that are currently in use (section 5.4), and for planned systems for the ELTs (section 5.5). Finally, we compare the increasing AO RTC complexities of modern systems (section 5.6).

5.1 General structure of AO real-time control systems

5.1.1 Architecture

The AO RTC has many tasks to perform, each may have different update frequencies. Its main hard real-time task is to receive a stream of pixel data from the WFS(s) and convert them into deformable mirror (DM) commands. There are two main steps to the processing. The first one is the wavefront processing, which generally involves extracting the wavefront slopes from the wavefront camera pixels. The next process, is to use this slope data to calculate the new positions for the actuators of the DM.

Figure 5.1 shows a simplified block diagram of an AO RTC loop processing chain. The hard real-time (time critical) tasks are inside the

bold dashed box. Other tasks which generally are auxiliary and run in parallel to the main AO loop are represented inside the dashed boxes. The items in the top dashed box represent soft real-time inputs to the RTC system such as the telescope telemetry. The items in the bottom dashed box represent soft real-time outputs such as real-time display, the optimisation of loop parameters, or the computation of offloads. These soft real-time processes are still time sensitive but are not as time critical as the hard real time tasks.

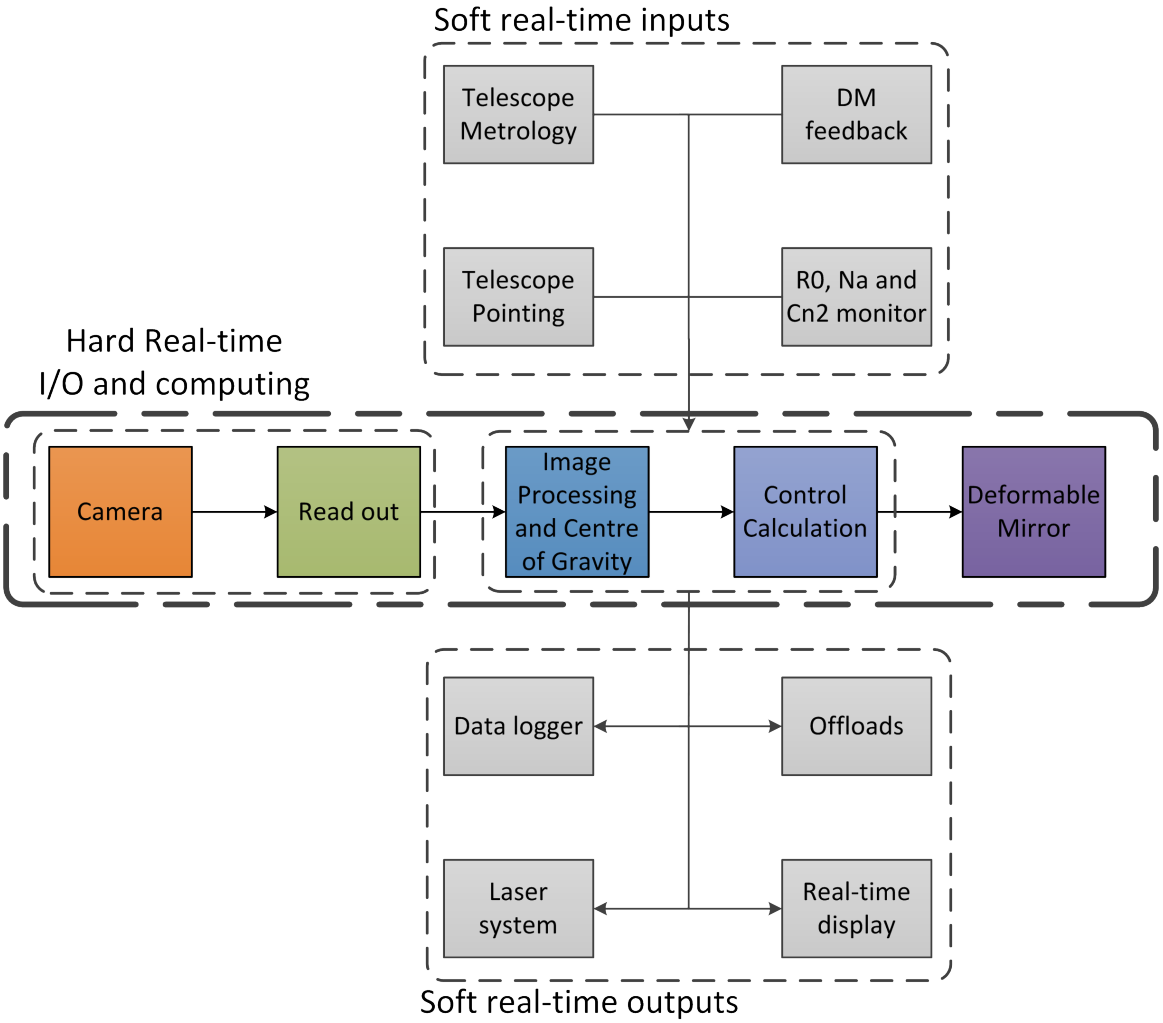


Figure 5.1: Simplified block diagram of an AO RTC loop processing chain and the association in the AO RTC.

The aim of this thesis is to identify and test many-core Commercial Off-The-Shelf (COTS) hardware that have the potential to reduce the

latency of the AO hard real-time loop. The hard real-time pipeline is not the only part of the AO RTC that requires powerful hardware. Many of the soft real-time tasks such as calibration and atmospheric estimations are also very computationally expensive.

The COTS technologies, which have been investigated for the hard real-time pipeline, may also be a good candidate to be used in the soft real-time computing. The Xeon Phi that is investigated in section 7 is a strong candidate for use in this way. However, this has not been investigated and is outside the scope of this thesis which focuses on the hard real-time pipeline.

5.1.2 Temporal considerations, chronogram

For a two-frame delay[1] AO system (typical of most AO instruments), the commands have to be sent to the DM before the next image from the camera has finished integrating. Figure 5.2 shows a simplistic timing diagram for the AO RTC control loop, using a frame-transfer CCD WFS. The processing stages (i.e. wavefront processing and DM control computation) are allowed to overlap. The processing of the wavefront images can start as soon as enough pixels have arrived at the wavefront processing unit (typically a row of sub-apertures), while more data is still being received. When the first few sub-apertures have been processed, the control calculation can begin using these sub-apertures. This overlap will help reduce the overall latency of the system.

[1] the two frame delay refers to the integration time of the camera and the calculation of the DM during the second frame.

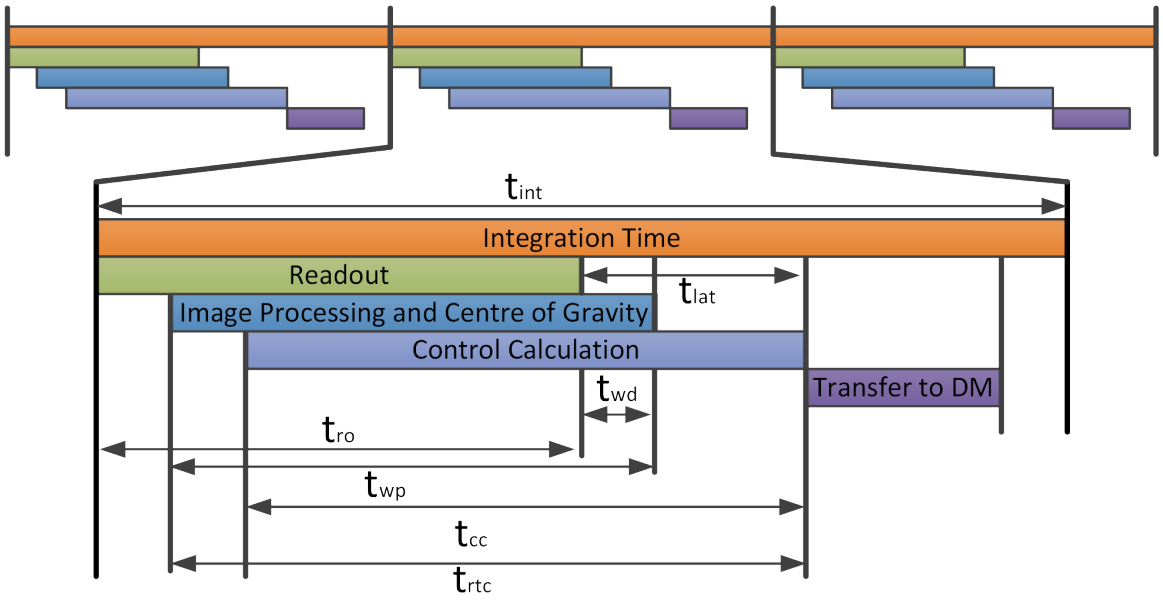


Figure 5.2: Overall AO control chain time diagram for a typical system, showing the overlapping processing steps.

Term	Symbol	Definition
WFS integration time	t_{int}	Time interval during which the WFS detector is exposed to light and accumulates (in other words integrates) signal to produce a sample of the wavefront error.
WFS read-out time	t_{ro}	Time elapsed between the instant at which the first pixel (or region) of the WFS detector is read-out and transmitted by the camera controller, and the instant at which the last pixel (or region) is read-out and transmitted by the camera controller.

continued ...

Term	Symbol	Definition
wavefront processing time	t_{wp}	Time elapsed between the first WFS camera pixel received at the RTC (or the WPU) and the last slope transmitted to the rest of the AO chain. This generally includes a calibration step and a center of gravity calculation.
wavefront processing delay	t_{wd}	Time elapsed between the last WFS camera pixel received at the RTC (or the wavefront processing unit) and the last slope transmitted to the rest of the AO chain.
Control calculation time	t_{cc}	Time elapsed between the first slope received at the RTC (or DM control unit), and the last DM command being transmitted by the RTC.
RTC computation time	t_{rtc}	Time elapsed between the first WFS camera pixel received at the RTC, and the last DM command being transmitted by the RTC. This accounts for both wavefront processing, and DM command computation time, which may or may not overlap in time.

continued ...

Term	Symbol	Definition
RTC latency	t_{lat}	Time elapsed between the last WFS camera pixel received at the RTC (or wavefront processing unit) and the last DM command data sent out by the RTC.

The AO chronogram presented above can be affected by the type of detector (see section 5.2.4.1) and the type of WFS (see section 5.2) used. For example, with a SH-WFS, computations can start as soon as a sufficient number of pixels have been read-out (typically a row of sub-apertures). For pyramid WFSs PYR-WFS, and depending on how the detector is read, one may have to wait until most or all of the pixels of the detector has been read-out.

5.2 Wavefront processing unit

Simply put, the function of the AO RTC loop is to take pixel data streamed from the wavefront camera, and convert it into DM commands. The wavefront sensor processing unit (WPU) is used to reduce the WFS frames to gradient vectors (slopes), which are then processed further to compute the DM commands via the wavefront reconstruction, typically a matrix-vector multiplication.

5.2.1 Shack-Hartman wavefront processing unit

Figure 5.3 shows a typical wavefront camera processing chain for a Shack-Hartmann WFS (SH-WFS), and is composed of two main processes: a calibration step (see section 5.2.2) and a processing step (see

section 5.2.3). Once the raw pixels have been read, they are calibrated with the dark map subtraction, flat fielding and background map subtraction.

In the second step, the process is somewhat different in the case of the SH-WFS and the PYR-WFS. But there are similarities in terms of complexity and computational load. This allows for WPU performance estimates to be applicable to both WFS types. For the SH-WFS, the processing involves a centre of gravity (CoG) calculation and the subtraction of the reference slopes. For the PYR-WFS and for each equivalent slope, the algorithm involves two additions, one subtraction, a division, and finally the subtraction of the reference slopes.

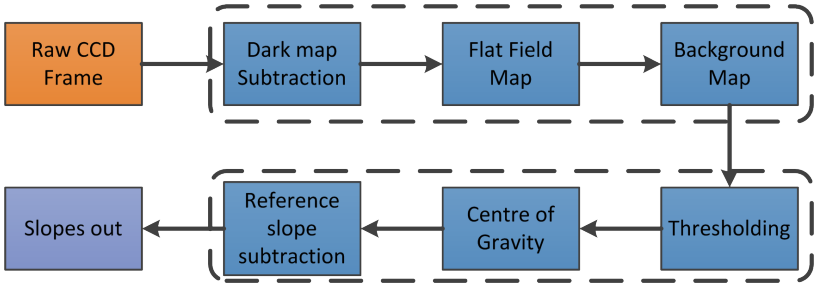


Figure 5.3: A typical processing chain for a Shack-Hartmann wavefront sensor. The calibration steps are shown in the upper dotted area and processing steps in the lower dotted area.

5.2.2 Wavefront sensor data reduction - pixel calibration

Data received from a camera will have residual artifacts caused by pixel-to-pixel sensitivity variations of the detector (i.e. flat fielding and background map). These calibrations techniques are generally common and are used on all detector types. They are used on systems from the largest telescopes to smart phone cameras that are carried in your pocket. Although the technique is common, each detector will require its own individual calibration created using these methods.

5.2.2.1 Flat field and dark map correction

Flat fielding and dark map subtraction are used to remove any variation in sensitivity between pixels.

The dark map subtraction attempts to remove any effects of dark current on the detector. Dark current is the residual current which may be flowing in a camera sensor when it is not illuminated due to thermal noise. An exposure is taken where the shutter is opened, but no light is allowed to hit the detector. The images are taken with the same exposure time and temperature as would be expected in normal operation. Since this is performed in darkness, only the energy from the CCD itself (i.e. the dark current) is present in the resultant image. This dark map (I_{Dark}) is subtracted from any images produced from the detector.

The second stage is the flat field map (I_{Flat}), which measures the response of each pixel in the CCD array to a uniform illumination. The resultant corrected image $I_{Corrected}$ can be calculated according to:

$$I_{Corrected} = (I_{Raw} - I_{Dark})/I_{Flat} \tag{5.1}$$

This pixel calibration involves a division which is very computationally demanding, much more so than a multiplication. For this reason, the fixed term $G = \frac{1}{I_{Flat}}$ is often pre-calculated leading to the following equation instead:

$$I_{Corrected} = (I_{Raw} - I_{Dark}) * G \tag{5.2}$$

5.2.2.2 Background map subtraction

An additional calibration step can be added, whereby a background map (I_{BACK}) is subtracted from the corrected image $I_{corrected}$. This section of the calibration enables the process to take into account any stray light that may impact the resultant image. The final image corrected from the dark map, the flat field and the background is calculated by:

$$I_{Corrected} = (I_{Raw} - I_{Dark}) * G - I_{Back} \tag{5.3}$$

5.2.3 Shack-Hartmann slope calculation methods

Once the image has been calibrated, it can be processed to extract the wavefront slopes. The calibration step is generally not WFS type specific, and is performed for most detector types. The wavefront processing however, is specific to the WFS type. The two main WFS that are currently being used are the PYR-WFS and the SH-WFS.

A sub-aperture of a SH-WFS using four pixels per sub-aperture is shown in figure 5.4 (typically more than 4 pixels per sub-apertures are used). The measurement of the displacement of the spot relative to the reference is used to calculate the local wavefront slope. Each sub-aperture will have a slope in X and in Y. This will give a slope vector that is twice the length of the number of sub-apertures.

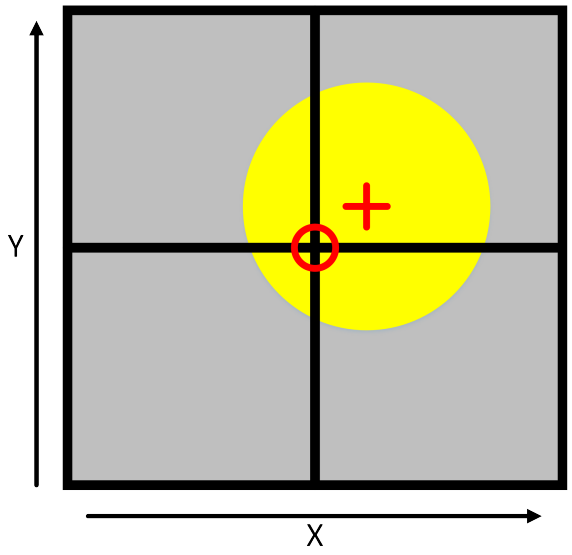


Figure 5.4: Illustration of the 2×2 sub-aperture. The yellow spot represents the light spot created by the associated lenslet. The red cross represents the current location of the CoG of the spot, and the red circle represents the position of the spot for no aberrations.

The calculation of the wavefront slopes from a SH-WFS image requires the estimation of sub-aperture spot displacements from detector pixel intensities. Position estimation is generally done by computing the spot's COG. It is the most widely used algorithm and is simplest to implement on a real-time computer. In order to reduce the impact of photon and detector noises other algorithms may be used. In this section, we will focus on the following algorithms:

- Centre of Gravity (COG)
- Weighted Centre of Gravity (WCOG)
- Matched Filter

5.2.3.1 Centre of Gravity

The CoG is the most widely used method of extracting the wavefront slopes from a SH-WFS image. This calculates the position of a spot by

applying the equation (5.4),

$$\widehat{x}_{COG} = \frac{\sum_{i,j} x_{i,j} I_{i,j}}{\sum_{i,j} I_{i,j}} \tag{5.4}$$

where $I_{i,j}$ is the pixel value at position (i, j) and $x_{i,j}$ the position coordinates of the CCD pixel (i, j). A similar equation can be obtained for the slopes in Y by simply replacing $x_{i,j}$ with $y_{i,j}$. The noise value of this method depends on the spot structure. In low flux, reducing the number of pixels per sub-aperture is necessary to minimise noise. At high-flux however, reducing the number of pixels per sub-apertures is not optimal, and other methods have therefore been developed. An alternative method adds a threshold to the pixels, putting any pixel value below this threshold to zero before the CoG calculation. This method is referred to as the thresholded centre of gravity (TCoG).

5.2.3.2 Weighted Centre of Gravity

The Weighted Centre of Gravity (WCoG) is similar to the CoG but adds weightings to each pixel in the sub-aperture depending on the flux[1]. The weighting function $W_{i,j}$ allows the attenuation of noisy low flux pixels. The slope location is calculated with equation (5.5).

There are two main implementations of the WCoG. A static version where the $W_{i,j}$ is set for each sub-aperture, and a dynamically WCoG where the $W_{i,j}$ is re-centred on the spot each frame[2].

$$\widehat{x}_{WCoG} = \frac{\sum x_{i,j} W_{i,j} I(i,j)}{\sum W_{i,j} I(i,j)} \tag{5.5}$$

In section 6.3 we present the results of these algorithms performed on the TILE-GX, where we focus on the static version of the WCoG.

5.2.3.3 Matched Filter

The matched filter algorithm for computing the wavefront local gradient is particularly suited for laser guide stars (LGS) wavefront sensing. It does this using equation (5.6),

$$\theta = R(I - I_0) \tag{5.6}$$

where R is the matched filter[3], I is the image pixel intensity and θ is the wavefront local gradient slopes (θ_x, θ_y). A full explanation of the algorithm can be found in (Gilles et al, 2006)[4].

5.2.4 Implications on hardware

5.2.4.1 Influence of detector type

The timing diagram of the AO control loop chain is mainly dictated by the WFS timing. This in turn depends on the individual detector type used. We can basically identify three main categories of detectors: CCD, CMOS and IR detectors.

For frame-transfer CCD devices, the integration and the read-out take place at different phases in time. First, all pixels are exposed during the same time t_{int}. They are then transferred with the shutter closed in a very short time. Finally, the pixels are read-out at a lower speed while a new integration takes place. It can be assumed that the detector pixels can be read-out in parallel, and that therefore the wavefront processing (or RTC computation) can start as soon as the first pixel (or in the case of a SH-WFS, the first row of sub-apertures) is read-out.

For CMOS detectors with a rolling shutter, the integration and read-out phases take place at the same time. Detector regions are read-out

in sequence and each region integrates during a time (t_{int}) which is equal to the time elapsed between two read-outs. It can be assumed that the pixels are read-out in parallel, and that therefore the wavefront processing can start as soon as the first region is read-out.

Finally, for IR detectors with Fowler sampling, it is assumed that the detector controller can only output pixels after some internal calculations. For Fowler sampling, the detector is reset in a short time. Non-destructive samples are taken immediately afterwards and towards the end of the integration time. The final pixel values are calculated based on these N measurements. This method is used to reduce read noise, and is reduced approximately by $\sqrt{N}$[5]. Therefore, the RTC computation cannot start before all pixels have been read-out.

5.2.4.2 Data transfer protocols

To move the pixel data from the WFS camera to the RTC many technologies can be used, each having different advantages and disadvantages. A summary of some of the main performance parameters is shown in table 5.2. The two main technologies used at the moment are Camera Link and GigE Vision.

Camera Link offers relatively high data rates with very low CPU usage and short cable lengths. This low CPU usage is obtained by allowing direct memory access (DMA) that puts camera data directly into memory, bypassing the CPU. The short cable lengths might cause problems for AO RTC s. This is due to the location of the computational hardware in relation to the instrument. The camera is located in the instrument on the telescope and usually the computational hardware is located away from the from the telescope in another room. This

means that long cable lengths (> 10 m) are required.

GigE Vision cameras use standard Ethernet cables to transfer the pixel data, traditionally 1 GbE but 10 GbE cameras are becoming available. These typically, offer lower data rates, but allows long cable lengths to be used. 10 GbE allow for higher data rates, while keeping the possibility of using long cables. However, the major downside to the this technology is the high amount of CPU time needed, though this can be mitigated with FPGA front ends or smart interconnects which are currently being developed. An advantage of the GigE Vision cameras is that they have the ability to multicast the data to multiple devices. This means that one device can focus on processing the incoming data while a second computer can record the data and/or control the camera, meaning that a single processing device does not have to handle multiple tasks. We have researched a technology that can efficiently use GigE Vision and 10 GbE in chapter 6.

Table 5.2: a comparison of the main camera bus read-out technologies.

Connect	Throughput (MBs^{-1})	Cable Length (m)	CPU usage
USB 3.0	640	15	high
Camera-Link	850	10	low
GigE Vision (1 GbE)	125	100	high
GigE Vision (10 GbE)	1250	100+	high

5.2.4.3 Impact of the WPU on the RTC

This wavefront processing step is very important in the RTC chain because it can reduce the amount of data the RTC needs to transfer and/or process by at least an order of magnitude. The hardware used in the wavefront processing unit needs to have two characteristics: high I/O to get the pixel data from the camera to the processors, and high

computational power to process the pixels. The chosen WPU hardware also needs to be able to function with the WFS camera format (e.g. Camera Link or GigE Vision).

I/O is an important consideration, since large amounts of data arrive onto the WPU at a very fast rate. In this regard, FPGAs and DSPs are good candidates due to the high performing I/O. CPUs can also be used as they generally support a large range of different interconnection technologies. GPUs on the other hand, without the use of GPU direct technology, do not have good I/O connectivity.

The other important consideration is the computing power that should be high enough to be able to compute the slopes in the required time. Although many technologies can provide the processing power necessary, problems with jitter, and/or variation is execution time are prevalent. The problem of variation in particular requires further investigation and testing as general purpose computational hardware does not tend to be real-time.

For SH-WFS, each sub-aperture can be processed independently of every other sub-aperture. This means that parallel (multi-core) hardware are a very efficient way of processing the data. Multi-core CPUs (such as the Xeon Phi) or GPUs (with the associated difficulty of uploading and downloading data into its memory, see paragraph 4.4.3) are good potential candidates for such applications.

5.3 Wavefront reconstruction

The next stage in the AO RTC loop is the wavefront reconstruction, sometimes called the control algorithm. This is where the wavefront slope vector is converted into DM commands.

5.3.1 Wavefront reconstruction algorithms

5.3.1.1 Matrix Vector Multiplication

The matrix-vector multiplication (MVM) is a simple control algorithm, that is the most commonly used in astronomy AO RTC systems. The MVM uses the slope vector s that has been calculated from the wavefront processing unit, which contains all the valid x and y slopes for each of the WFS sub-apertures. The slope vector is equal to:

$$s = D\phi \tag{5.7}$$

ϕ is the incoming wavefront and D is the interaction matrix. The interaction matrix, sometimes known as the poke matrix, is a matrix that converts DM commands into WFS measurements and is measured during the system calibration. A simple way of creating it is by poking each actuator in turn, and recording the measured slopes on the WFS(s).

To estimate the phase $\hat{\phi}$ from the slope measurements we need to calculate the inverse of D, D^{+}. Since D is generally rectangular and not square, this cannot be inverted through classical techniques. To invert non-square matrices, a pseudo-inversion is performed. There are many techniques to do this such as a singular value decomposition. An estimate of the incoming phase can be calculated using the measured slopes by:

$$\hat{\phi} = D^{+}s \tag{5.8}$$

This method has been used extensively from the earliest AO RTC systems (it was, for example, used on COME-ON[6]), and is the most

widely used control algorithm today. This approach provides good results and is conceptually the simplest control algorithm. The computational requirements of MVM increases with a factor of n^2, where n is the number of actuators of the DM, so for large systems this method will have very large computational loads.

5.3.1.2 Alternative reconstruction algorithms

CuReD

To reduce the complexity load of the MVM for large systems, other algorithms have been developed. One of the least computationally intense algorithms is called a Cumulative Reconstructor with domain Decomposition (CuReD). This algorithm is conceptually quite simple and based on the Cumalative Reconstructor (CuRe).

CuRe is an algorithm to reconstruct the wavefront by summing up the slopes from a SH-WFS to create the wavefront surface. This assumes a continuous wavefront that has no discontinuities. Each sub-aperture slope will connect to the slope on either side. An illustration of this can be shown in figure 5.5. Since there can be no discontinuities, the third section has to be moved up to connect with the previous sub-aperture. This technique is then used across the whole wavefront in both axes. This wavefront can then be applied to the DM using a MVM to calculate the voltages, though it has been shown that you can apply this shape straight to the DM if the system is very well aligned[7].

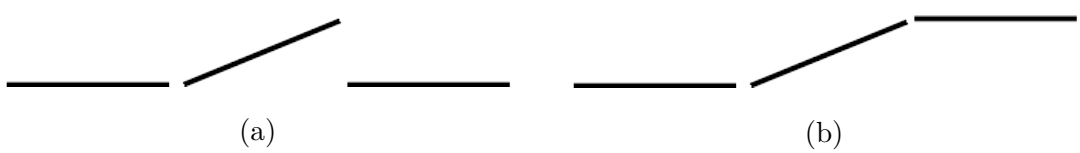


Figure 5.5: Illustration of reconstructing a continuous wavefront surface from 3 independent slope measurement.

CuRe has been shown to have high levels of noise propagation, when used with systems with a high number of sub-apertures. To solve this issue, CuRe has been adapted to use domain decomposition to create CuReD. This splits the wavefront into domains where CuRe can be applied, which reduces propagated noise across the whole sub-aperture[8]. This method increases in complexity with $\mathcal{O}(n)$, and has a computational load of 20 n and is able to be parallelisable.

Fourier transform reconstructor

The Fourier transform reconstructor (FTR) was proposed in 1986[9]. This technique uses the Fourier basis set to reconstruct the wavefront (for more information see in particular[10]).

This technique is more complex to understand than the MVM model, though it provides a more computationally efficient method. Computationally this model increase complexity with a factor of $\mathcal{O}(n \log n)$, which for large systems with high number of degrees of freedom, allows it to be more computationally efficient, and requires four and a half times less computation when compared to MVM[11]. In addition, this method takes advantage of Fourier basis sets. These are well established signal processing techniques and hardware which can be used, such as DSPs.

Fractal Iterative Method

Unlike the other methods presented above, the FRactal Iterative Method (FRIM) is iterative[12]. It uses the knowledge that the atmospheric turbulence has a power law relation within the range of the inner and outer scale ($l_0 < x < L_0$). FRIM has a computational complexity of $\mathcal{O}(n)$[13].

5.3.1.3 Algorithm comparison

Table 5.3 gives a summary of a selection of algorithms and the associated computational complexity; this is not an exhaustive list and other algorithms do exist. The current direction is for computational hardware to have more and more cores. Algorithms that are easily parallelised, such as the MVM, can therefore take full advantage of the evolution of modern hardware. Other algorithms such as FRIM, which require iterative processes will make optimisation (i.e. taking full adavantage of the multi-core nature of modern processors) much more difficult (if at all possible).

Table 5.3: Comparison of various wavefront reconstruction algorithms and their complexity.

Method	Complexity	Operations
MVM	$\mathcal{O}(n^2)$	
CuRed	$\mathcal{O}(n)$	20 N
FTR	$\mathcal{O}(n \log n)$	
FRIM	$\mathcal{O}(n)$	350 N

We have chosen to investigate MVM in particular for a number of reasons. It is the standard on the majority of AO systems and is the current baseline the vast majority of AO systems to be installed on ELTs. It is a simple algorithm to understand and allows for simple numerical testing to compare results (and intermediate calculation steps) between the intended mathematical control and the actual implementation on the RTC. The aim of this investigation is not to demonstrate new algorithms but to accelerate the current baseline with new technology. For these reasons we have made the decision to target the MVM algorithm only and have specifically chosen hardware that can take advantage of

the parallelisable nature of the algorithm.

5.3.2 Temporal control

Once the control vector is calculated, and mainly for reasons of stability, the actual control vector sent to the DM needs to be regularised. In classical AO systems, a proportional integrator (PI) is typically used. The control of a PI, in the discrete case can be expressed as:

$$u_t = K_p * e_t + K_i \sum_{n=1}^{k} e_n \tag{5.9}$$

Where u_t is the control vector sent to the DM at discrete time t and e_t is the resultant vector from the wavefront reconstruction (MVM) which is multiplied by the gain (K_p). This is the proportional term in the PI controller. The next term ($K_i \sum_{n=1}^{k} e_n$) is referred to as the integrator, as it integrates over the previous k inputs, K_i is the integral gain. This is why this type of control is know as a PI controller. This is more commonly shown in the velocity form:

$$u_t = u_{t-1} + K_p(e_t - e_{t-1}) + K_i(e_t) \tag{5.10}$$

The advantage of this is that we no longer need to keep track of the summation from (5.9). This also adds little complexity to the overall RTC system in comparison to the wavefront reconstruction using an MVM. In this regard, we are not considering this in the testing of the hardware in the later chapters of this thesis.

5.4 Example of AO RTC systems

This section briefly covers some of the AO RTC that have been used in the recent past, or are currently in operation in large observatories.

5.4.1 NAOS

Nasmyth Adaptive Optics System (NAOS) was the first AO system on the VLT 8 m telescope, beginning development in 1999. It has two SH-WFS. The first operates at visible wavelengths and the second operates in the infrared. Each WFS has two lenslets arrays, which can be switched. The first lenslet array is a 14×14 (144 valid sub-aperture[2]), and the second 7×7 (36 valid sub-apertures). This configuration is used to allow the WFSs to operate over a large magnitude range. The larger lenslet array is designed to sample bright natural guide stars while the smaller one is designed for use with faint natural guide stars[14]. The wavefront correction is provided by a tip-tilt beam steering mirror, and an 185 actuator continuous faceplate DM.

The system is controlled by the RTC, which is based on four modular boards using a modified C40 processor from Texas Instruments. The C40 is a DSP that can operate at 60 MFLOPs.

It uses a classical MVM reconstruction algorithm. This system can be updated at a maximum frequency of 600 Hz[15]. Table 5.4 shows a summary of the NAOS instrument. This can also be seen in Table 5.8 with a comparison to other AO instruments.

Table 5.4: Summary of the main characteristics of NAOS. WPU and WFR, give the hardware the wavefront processing and reconstruction respectively.

	AO	WFS (lenslets)	DM (actuators)	Freq (Hz)	WPU	WFR
NAOS	SCAO	14×14	185	600	DSP	DSP

[2]By valid, we refer to only the sub-aperture illuminated by the telescope, (see appendix A.2).

5.4.2 SPARTA

Standard Platform for Adaptive Optics Real-time applications (SPARTA), created by the European Southern Observatory is a AO RTC toolbox to speed the development of AO RTCs for the VLT. The aim of SPARTA is to provide a standard platform to build and test AO RTCs[16].

The SPARTA RTC consists of a real-time box and a co-processing cluster. The real-time box contains the hard real-time pipeline, wavefront processing, reconstruction and control, and offloads all non critical real-time tasks to the co-processing cluster. The real-time path of SPARTA is performed by a hybrid system of FPGAs and DSPs to produce the lowest latency response possible. The co-processing cluster is a set of multi-CPU, multi-core Linux servers interconnected on a private real-time Ethernet network.

Figure 5.6 shows the architectural schematic for SPARTA. The RTC pipeline is provided by FPGAs, DSPs and powerPCs, while all non-real-time tasks are performed on the co-processor servers. The pixels from the wavefront cameras are fed into VPF1 boards, which contain two virtex-II pro FPGAs being controlled by power PCs. A single VPF1 board can provide the WPU for two CCD240 SH-WFS (two boards are needed for AOF systems). The slopes are then transferred to BittWare T2v6 boards. These boards have 8 DSPs which provide the wavefront reconstruction processes. One T2v6 board is needed per WFS to provide the required processing power. SPHERE uses two to allow for the interaction matrix to be updated without impacting the performance. The wavefront reconstruction results are then sent to a final VPF1 board which implements the integrator with optimised modal gain controller, checks for any saturation of commands as well as

provides the anti-vibration control. The real-time pipeline data transfer is provided by a VXS backplane running at 2.5 Gbit s^{-1} via a zero-latency VXS switch, using the sFPDP protocol. The real-time box connects to the co-processing servers using Gigabit Ethernet.

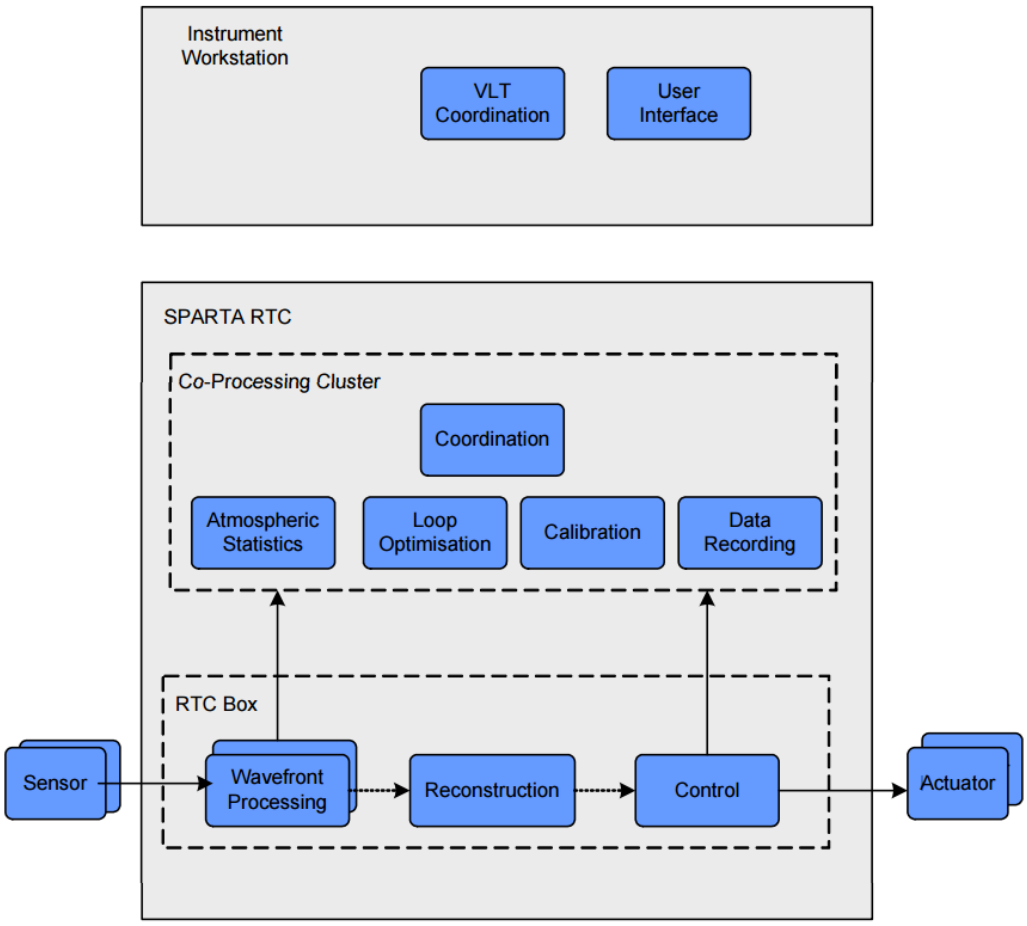


Figure 5.6: SPARTA architecture schematic Copyright: ESO[16]

Table 5.5 gives a brief overview of two of the main systems that use SPARTA as the RTC. We present SPHERE, which is the current planet-finding instrument on the VLT[17] with an extreme AO system[18]. SPARTA is also in use with VLT AOF[19] instruments GRAAL[20] and GALACSI[21], and is planned for use with ERIS[22].

Table 5.5: AO instruments that use SPARTA as the RTC. WPU and WFR, give the hardware the wavefront processing and reconstruction respectively.

	AO	WFS (lenslets)	DM (actuators)	Freq (Hz)	WPU	WFR
SPHERE	XAO	40×40	1377	1500	FPGA	DSP
AOF	LTAO	4×40×40	1170	1000	FPGA	DSP

As we move from the VLT to the era of the ELTs, the current iteration of SPARTA will not be able to cope with the increase in computational complexity:

> *the E-ELT with its instruments poses new challenges in terms of cost and computational complexity. Simply scaling the current SPARTA implementation to the size of E-ELT AO system would be unnecessary expensive and in some cases not even feasible.[23]*

5.4.3 DARC

Durham Adaptive Optics Real-time Control (DARC) is a RTC designed to be a highly customisable generic AO RTC software. It was originally designed to run and make use of multi-core CPU architectures, but it is now able to be used with more complex heterogeneous computing architectures[24]. A large amount of development has been put into the use of hardware accelerators such as FPGA and GPUs[25]. DARC also supports many different system architectures, as well as supporting many modes of AO such as MOAO and GLAO. It has also been developed to be able to perform many different wavefront reconstruction algorithm, such as MVM, learn-and-apply, and CuReD. These have

also been tested on sky[7, 26]. DARC also supports many different WFS designs such as SH-WFS and PYR-WFS.

DARC has been used as the RTC for CANARY, a MOAO demonstrator for EAGLE[3], and has being tested on the 4.3 m William Herschel Telescope in the Canary Islands[27]. CANARY (shown in figure 5.4.3) has four 7×7 SH-WFS, and each sub-aperture has 16×16 pixels and is imaged on a 128×128 detector. The correction is made on a 8×8 piezostack DM conjugated to the pupil plane. The RTC runs at a frequency of 250 Hz with a 800 μs latency. The real-time pipeline uses hardware based FPGA WPUs similar to those used in SPARTA[28]. The slopes are then sent via a Serial-FPDP interface, to the wavefront reconstruction of DARC is processed on multi-core CPU based hardware[29].

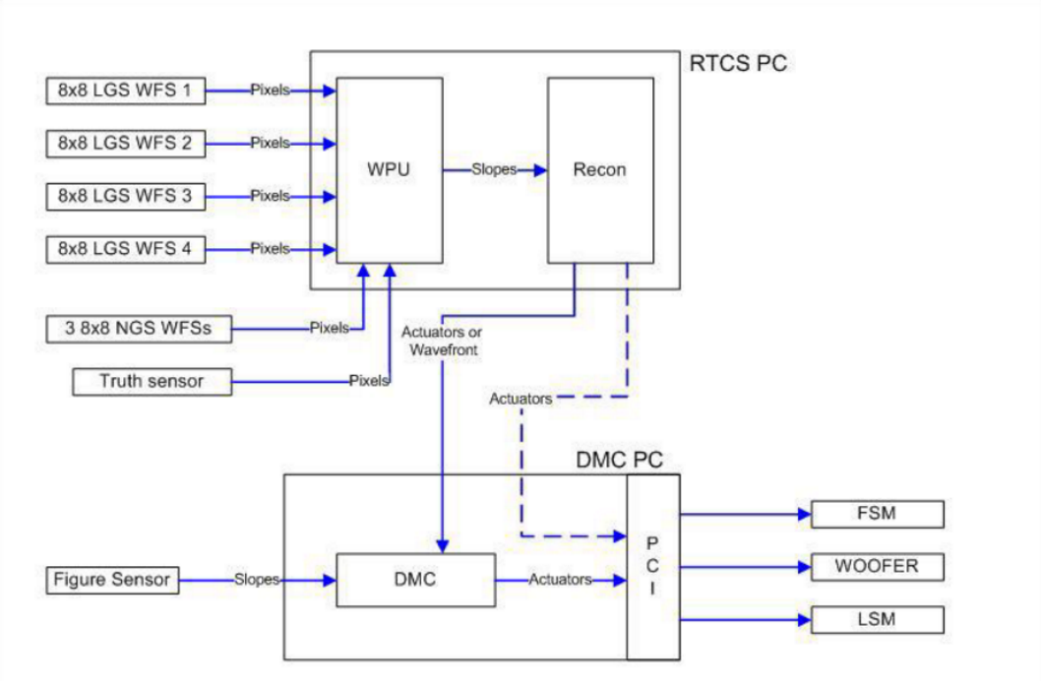


Figure 5.7: Block diagram of CANARY Phase B RTC configuration[30].

A new project called the Canary-Hosted Upgrade for High-Order Adaptive Optics is, as the name suggests, a high-order upgrade to

[3]EAGLE has since been merged with EVE to create MOSAIC

Canary[31]. This system will be using a 31×31 MEMS DM in addition to the smaller DM already hosted in Canary[32]. CHOUGH will use the same RTC as Canary but it will be upgraded to take advantage of high performance hardware accelerators such as Xeon Phis or GPUs[33].

We have used DARC intensively, in particular for testing the performance of the Xeon Phi (see chapter 7).

Table 5.6: Summary of CANARY and CHOUGH. WPU and WFR, give the hardware the wavefront processing and reconstruction respectively.

	AO	WFS (lenslets)	DM (actuators)	Freq (Hz)	WPU	WFR
CANARY	LTAO	4×(7×7)	52	250	FPGA	CPU
CHOUGH	LTAO	30×30	950	1000	FPGA	TBC (CPU/GPU)

5.5 Example of planned AO RTC systems

In this section, we intend to discuss a few examples of the planned RTC systems being developed for the ELTs. This section is not exhaustive, but intends to give the reader a sense of the on-going projects in the field.

5.5.1 NFIRAOS

The Near Field InfraRed Adaptive Optics System (NFIRAOS) is planned to be the AO facility for the Thirty Meter Telescope. It is planned to offer the required AO correction needed for three science instruments on the TMT. NFIRAOS is an MCAO system using six laser guide stars, five in a pentagon pattern and one on axis, as well as a single natural guide star.

The baseline for the NFIRAOS control system is to use Xeon E5 CPUs, although GPUs[34] and Xeon Phis[35] have also been investigated. With the recent release of Xeon Phi Knights Landing (the second generation of commercial Xeon Phis), this might be subject to change as the advertised specifications will allow the calculations to run on cheaper hardware, using less power and taking up less server rack space. In terms of complexity, the tomographic reconstruction is performed by a 35k×8k classical MVM at 800 Hz.

Table 5.7: Summary of the main characteristics of NFIRAOS.

	AO	WFS (lenslets)	DM (actuators)	Freq (Hz)	Hardware
NFIRAOS	MOAO	6×(60×60)	7673	800	CPU/Xeon Phi

An architectural overview of NFIRAOS is presented in figure 5.8. Each WFS has its own high-order processing (HOP) server to process the incoming wavefront slopes, and a wavefront corrector control server to calculate the DM commands.

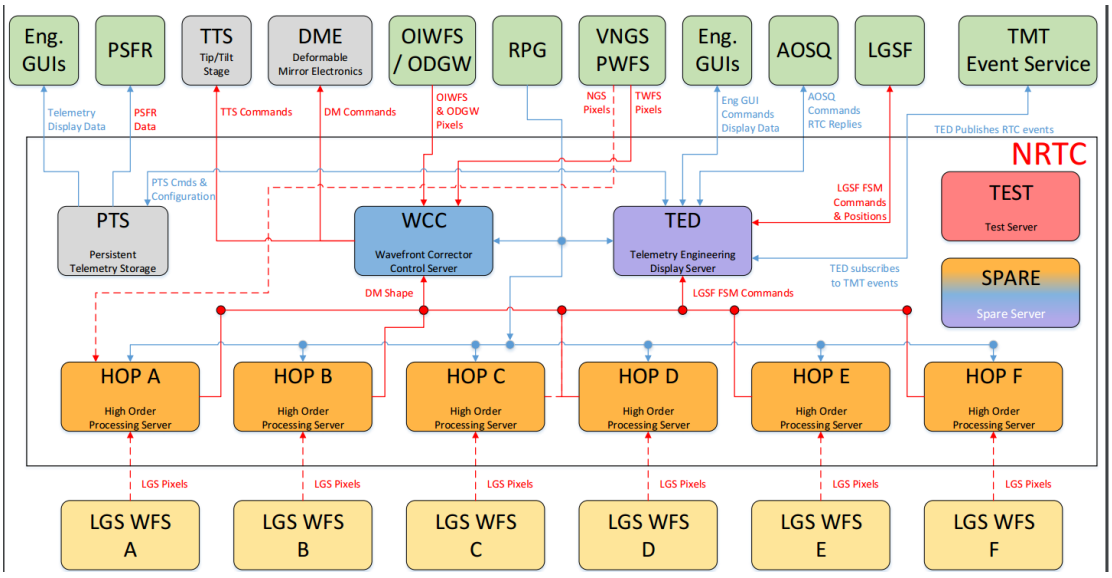


Figure 5.8: NFIRAOS architecture schematic. copyright: TMT (TMT.AOS.PRE.16.024.REL01)

5.5.2 Green FLASH

The aim of the Green FLASH project is to design and build an AO RTC prototype which is capable of providing the required processing for E-ELT first light instruments[36]. Green FLASH is trying to bridge the gap between VLT scale and E-ELT scale AO RTCs. This project's primary objective is to produce a prototype cluster which is able to reach a sustained performance of 1.5 TMAC s^{-1} of computing power, whilst processing 250 Gbit s^{-1} of streaming data with a maximum jitter of 100 μs over 1 s of operation, using COTS accelerators such as GPUs as compute engines. It shall be compatible with high performance switch solutions, based on standard serial protocols (TCP/UDP through 10 G Ethernet and 40 G Infiniband).

The performance of the prototype cluster will be assessed by measuring the performance of the Cholesky factorisation[4], the upload of the control matrix and matrix-vector computation. These calculations should be performed with minimal introduced latency and jitter in the control process.

An example block diagram for Green FLASH project is presented in figure 5.9

[4]This is a technique to calculate the pseudo-inverse of a non-square matrix. The result of which is needed when computing a new control matrix.

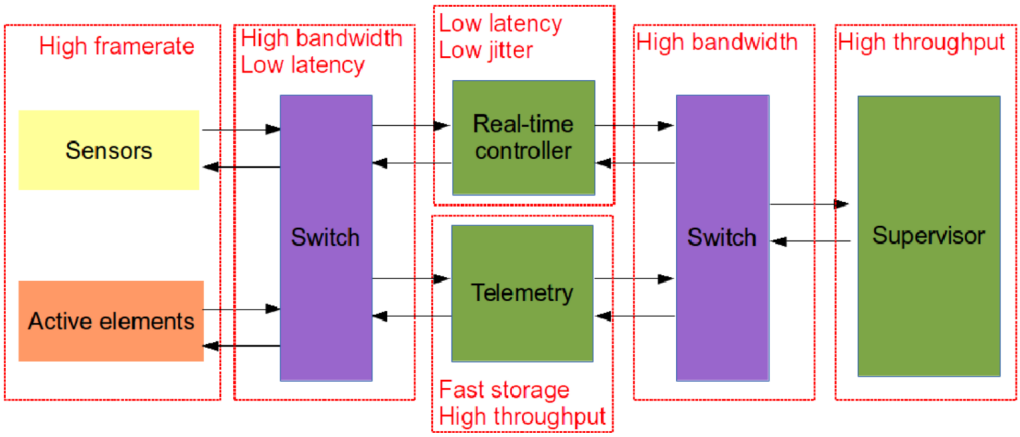


Figure 5.9: Green Flash Architecture Schematic. copyright: Gratadour[37].

Green FLASH is still in the beginning stages of the project and is not mature enough to have a complete design, but aims at designing the building blocks for a system that can be developed for the E-ELTs first light instrument, such as MAORY[38].

5.6 Complexity of E-ELT RTC systems

We have presented a selection of current and future AO systems[5]. Figure 5.10 illustrates the increase in computational complexity of these systems over time. The MACS values of each of these systems have been estimated assuming these systems are using SH-WFS, the WCoG algorithms and the wavefront reconstruction is performed using a MVM. As no additional calculations have been assumed, the actual complexity value is likely to be higher. A blue shaded section has been added to indicate the period of time over which the work presented in this thesis was performed.

[5]The data in this section is taken from these systems.

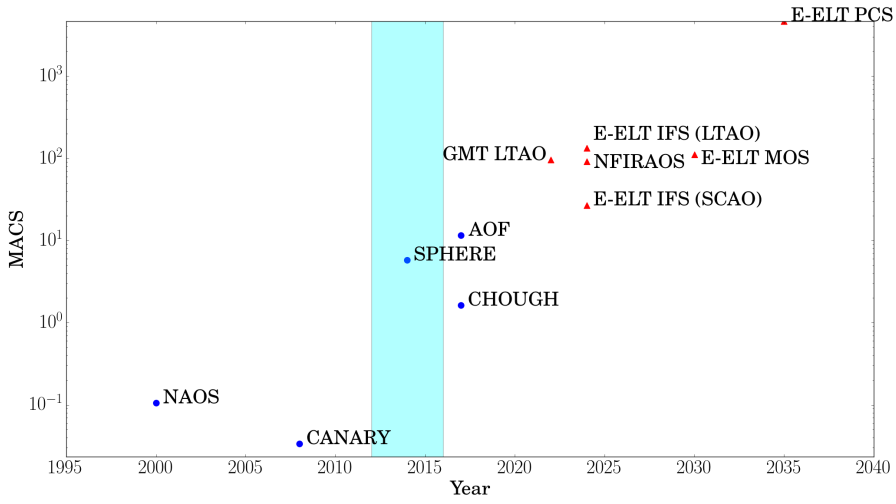


Figure 5.10: The increasing complexity of AO systems over time. The shaded blue section refers to the years the work within this thesis has taken place.

Table 5.8 gives a more detailed breakdown of the major systems presented in figure 5.10. All these current RTCs are using some non-CPU hardware in the hard RTC pipelines. These systems are making use of the deterministic nature of FPGAs and the power of DSPs. Plans for the future AO RTCs, indicate that higher numbers of RTCs are using more generic hardware such as CPUs (NFIRAOS) and GPUs.

Table 5.8: Summary of the main characteristics of current and future AO systems. WPU and WFR, give the hardware the wavefront processing and reconstruction respectively.

Current telescopes						
	AO	WFS (lenslets)	DM (actuators)	Freq (Hz)	WPU	WFR
NAOS	SCAO	14×14	185	600	DSP	DSP
CANARY	LTAO	4×(7×7)	52	250	FPGA	CPU
SPHERE	XAO	40×40	1377	1500	FPGA	DSP
AOF	LTAO	4×(40×40)	1170	1000	FPGA	DSP
CHOUGH	SCAO	30×30	950	1000	FPGA	CPU/GPU
Future ELTs						
	AO	WFS (lenslets)	DM (actuators)	Freq (Hz)	Hardware	
E-ELT IFS	LTAO	6×(74×74)	5316	500	TBC	
NFIRAOS	MOAO	6×(60×60)	7673	800	CPU/Xeon Phi	
E-ELT MOS	MOAO	10×(74×74)	5316	250	TBC	

The systems presented for the ELTs are all mainly first generation AO systems, and the baseline for the ELTs. In the future these AO systems are only going to be more complex and require more powerful RTCs. As has been stated earlier, the aim of this thesis to investigate novel many-core technologies that may be suitable for these instruments. As there are differences in all these instruments we are taking a general approach to our testing. Our intent being that the result presented in later chapters can useful for building any AO RTC. Although the aim is to remain general, we are still focusing our testing on these ELT baseline instruments.

5.7 Conclusion

In this chapter, we have discussed real-time computing and its roles in AO instruments. We have presented the main modules that constitute the most important building blocks of an AO RTC: the wavefront processing unit and the wavefront reconstruction unit. Both modules are crucial elements to ensure that the overall latency, and therefore the temporal error, are minimised. We have presented current and future RTC systems and the associated complexity.

References

[1] M Nicolle, T Fusco, G Rousset, and V Michau. Improvement of shack–hartmann wave-front sensor measurement for extreme adaptive optics. *Optics letters*, 29(23):2743–2745, 2004.

[2] S Thomas, T Fusco, A Tokovinin, M Nicolle, V Michau, and G Rousset. Comparison of centroid computation algorithms in a shack–hartmann sensor. *Monthly Notices of the Royal Astronomical Society*, 371(1):323–336, 2006.

[3] L Gilles and BL Ellerbroek. Constrained matched filtering for extended dynamic range and improved noise rejection for shack-hartmann wavefront sensing. *Optics letters*, 33(10):1159–1161, 2008.

[4] Luc Gilles and Brent Ellerbroek. Shack-hartmann wavefront sensing with elongated sodium laser beacons: centroiding versus matched filtering. *Applied optics*, 45(25):6568–6576, 2006.

[5] G Smadja, C Cerna, A Castera, and A Ealet. Frequency analysis of the noise in the fowler (n) sampling of a h2rg (2k× 2k) near-ir detector. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 622(1):288–294, 2010.

[6] Corinne Boyer, Vincent Michau, and Gerard Rousset. Adaptive optics: interaction matrix measurements and real time control algorithms for the come-on project. In *The Hague'90, 12-16 April*, pages 63–81. International Society for Optics and Photonics, 1990.

[7] Urban Bitenc, Alastair Basden, Nazim Ali Bharmal, Tim Morris,

Nigel Dipper, Eric Gendron, Fabrice Vidal, Damien Gratadour, Gérard Rousset, and Richard Myers. On-sky tests of the cured and hwr fast wavefront reconstruction algorithms with canary. *Monthly Notices of the Royal Astronomical Society*, 448(2):1199–1205, 2015.

[8] Matthias Rosensteiner. Wavefront reconstruction for extremely large telescopes via cure with domain decomposition. *J. Opt. Soc. Am. A*, 29(11):2328–2336, Nov 2012.

[9] Klaus R Freischlad and Chris L Koliopoulos. Modal estimation of a wave front from difference measurements using the discrete fourier transform. *JOSA A*, 3(11):1852–1861, 1986.

[10] Lisa A Poyneer, Donald T Gavel, and James M Brase. Fast wavefront reconstruction in large adaptive optics systems with use of the fourier transform. *JOSA A*, 19(10):2100–2111, 2002.

[11] Lisa A Poyneer, Mitchell Troy, Bruce Macintosh, and Donald T Gavel. Experimental validation of fourier-transform wavefront reconstruction at the palomar observatory. *Optics letters*, 28(10):798–800, 2003.

[12] Clémentine Béchet, Michel Tallon, and Eric Thiébaut. Frim: minimum-variance reconstructor with a fractal iterative method. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 62722U–62722U. International Society for Optics and Photonics, 2006.

[13] Michel Tallon, Isabelle Tallon-Bosc, Clémentine Béchet, Fabien Momey, Marie Fradin, and Éric Thiébaut. Fractal iterative method for fast atmospheric tomography on extremely large telescopes. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 77360X–77360X. International Society for Optics and Photonics, 2010.

[14] P Fautrier, G Rousset, RJ Dorn, C Cavadore, J Charton, C Cumani, T Fusco, N Hubin, P Kern, JL Lizon, et al. Performances and results on the sky of the naos visible wavefront sensor. *Adaptive Optical System Technology II*, 4839.

[15] Rainer Lenzen, Markus Hartung, Wolfgang Brandner, Gert Finger, Norbert N Hubin, Francois Lacombe, Anne-Marie Lagrange, Matthew D Lehnert, Alan FM Moorwood, and David Mouillet. Naos-conica first on sky results in a variety of observing modes. In *Astronomical Telescopes and Instrumentation*, pages 944–952. International Society for Optics and Photonics, 2003.

[16] Enrico Fedrigo, Reynald Bourtembourg, Robert Donaldson, Christian Soenke, Marcos Suarez Valles, and Stefano Zampieri. Sparta for the vlt: status and plans. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 77362I–77362I. International Society for Optics and Photonics, 2010.

[17] Jean-Luc Beuzit, Markus Feldt, Kjetil Dohlen, David Mouillet, Pascal Puget, Francois Wildi, Lyu Abe, Jacopo Antichi, Andrea Baruffolo, Pierre Baudoz, et al. Sphere: a'planet finder'instrument for the vlt. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 701418–701418. International Society for Optics and Photonics, 2008.

[18] J-F Sauvage, T Fusco, C Petit, S Meimon, E Fedrigo, M Suarez Valles, M Kasper, N Hubin, J-L Beuzit, J Charton, et al. Saxo, the extreme adaptive optics system of sphere: overview and calibration procedure. In *SPIE Astronomical Telescopes+ Instrumen-*

tation, pages 77360F–77360F. International Society for Optics and Photonics, 2010.

[19] R Arsenault, P-Y Madec, N Hubin, J Paufique, S Stroebele, C Soenke, R Donaldson, E Fedrigo, S Oberti, S Tordo, et al. Eso adaptive optics facility. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 701524–701524. International Society for Optics and Photonics, 2008.

[20] Jérome Paufique, J Argomedo, R Arsenault, R Conzelmann, R Donaldson, N Hubin, L Jochum, A Jost, M Kiekebusch, J Kolb, et al. Status of the graal system development: very wide-field correction with 4 laser guide-stars. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 844738–844738. International Society for Optics and Photonics, 2012.

[21] S Ströbele, P La Penna, R Arsenault, RD Conzelmann, B Delabre, M Duchateau, R Dorn, E Fedrigo, N Hubin, J Quentin, et al. Galacsi system design and analysis. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 844737–844737. International Society for Optics and Photonics, 2012.

[22] A Riccardi, S Esposito, G Agapito, J Antichi, V Biliotti, C Blain, R Briguglio, L Busoni, L Carbonaro, G Di Rico, et al. The eris adaptive optics system. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 99091B–99091B. International Society for Optics and Photonics, 2016.

[23] Enrico Fedrigo and Robert Donaldson. Sparta roadmap and future challenges. In *SPIE Astronomical Telescopes+ Instrumenta-*

tion, pages 77364O–77364O. International Society for Optics and Photonics, 2010.

[24] Alastair Basden, Deli Geng, Richard Myers, and Eddy Younger. Durham adaptive optics real-time controller. *Applied optics*, 49(32):6354–6363, 2010.

[25] A. G. Basden and R. M. Myers. The durham adaptive optics real-time controller: capability and extremely large telescope suitability. *Monthly Notices of the Royal Astronomical Society*, 424(2):1483–1494, 2012.

[26] Alastair Basden, Urban Bitenc, and David Jenkins. Novel algorithm implementations in darc: the durham ao real-time controller. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 99094R–99094R. International Society for Optics and Photonics, 2016.

[27] E Gendron, F Vidal, M Brangier, T Morris, Z Hubert, A Basden, G Rousset, R Myers, F Chemla, A Longmore, et al. Moao first on-sky demonstration with canary. *Astronomy & Astrophysics*, 529:L2, 2011.

[28] Alastair Basden, Richard Myers, Nigel Dipper, and Tim Morris. Real-time control developments for the canary moao instrument at durham. In *Second AO4ELT Conference. AO4ELT2*, 2011.

[29] Timothy J Morris, Alastair G Basden, Fabrice Vidal, Andrew P Reeves, Eric Gendron, Richard M Myers, Zoltan Hubert, Edward J Younger, Andy Longmore, Matthieu Cohen, et al. Tests of open-loop lgs tomography with canary. In *SPIE Astronomical Tele-*

scopes+ Instrumentation, pages 84470K–84470K. International Society for Optics and Photonics, 2012.

[30] Richard M Myers, Zoltán Hubert, Timothy J Morris, Eric Gendron, Nigel A Dipper, Aglaé Kellerer, Stephen J Goodsell, Gérard Rousset, Eddy Younger, Michel Marteaud, et al. Canary: the on-sky ngs/lgs moao demonstrator for eagle. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 70150E–70150E. International Society for Optics and Photonics, 2008.

[31] Daniel Hölck, Nazim Ali Bharmal, Cornelis M Dubbeldam, and Richard M Myers. Chough: spatially filtered shack-hartmann wave-front sensor for hoao. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 990930–990930. International Society for Optics and Photonics, 2016.

[32] Nazim A Bharmal, Alastair G Basden, Cyril J Bourgenot, Martin Black, Cornelis M Dubbledam, David M Henry, Daniel Hölck-Santibanez, Timothy J Morris, David J Robertson, Jürgen Schmoll, et al. Chough: implementation and performance of a high-order 4m ao demonstrator. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 990948–990948. International Society for Optics and Photonics, 2016.

[33] Bharmal Nazim, Daniel Holck, Richard Myers, Timothy Morris, Marc Dubbledam, Alastair Basden, and Edward Younger. Progress with the 4m high-order ao demonstrator, chough. In *Adaptive Optics for Extremely Large Telescopes 4–Conference Proceedings*, volume 1, 2015.

[34] Lianqi Wang. Design and testing of gpu based rtc for tmt nfiraos. In *Third AO4ELT Conference. AO4ELT3*, volume 13172, 2013.

[35] Malcolm Smith, Dan Kerley, Glen Herriot, and Jean-Pierre Véran. Benchmarking hardware architecture candidates for the nfiraos real-time controller. volume 9148, page 91484K, 2014.

[36] D Gratadour, N Dipper, R Biasi, H Deneux, J Bernard, J Brule, R Dembet, N Doucet, F Ferreira, E Gendron, et al. Green flash: energy efficient real-time control for ao. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 99094I–99094I. International Society for Optics and Photonics, 2016.

[37] D. Gratadour. A real-time control computer for the e-elt, January 2016.

[38] Laura Schreiber, Emiliano Diolaiti, Carmelo Arcidiacono, Andrea Baruffolo, Giovanni Bregoli, Enrico Cascone, Giuseppe Cosentino, Simone Esposito, Corrado Felini, Italo Foppiani, et al. Dimensioning the maory real time computer. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 99094L–99094L. International Society for Optics and Photonics, 2016.

Chapter 6

Wavefront processing unit: an I/O problem

Part of this work has been presented and published at the Adaptive optics for ELTs 4 in 2015 and SPIE astronomical telescopes and instrumentation 2016:

- *David Barr, Alastair Basden, Nigel Dipper, and Noah Schwartz. Reducing adaptive optics latency using many-core processors. Proc. AO4ELT4, 1, 2015.*
- *David Barr, Noah Schwartz, Andy Vick, John Coughlan, Rob Halsall, Alastair Basden, and Nigel Dipper. "Novel technology for reducing wavefront image processing latency." SPIE Astronomical Telescopes+ Instrumentation. International Society for Optics and Photonics, 2016*

Contents

In an adaptive optics (AO) system, the role of the wavefront sensor processing unit (WPU) can be briefly summarised as to enable the reduction of the wavefront sensor (WFS) data to gradient vectors (i.e. slopes data that is then passed further to the wavefront reconstruction unit). We will be focusing in this chapter on the processing required for the Shack-Hartmann WFS (SH-WFS). The image is split into regions called sub-apertures, which can be processed independently from each other. A detailed explanation of the role of the WPU and the algorithms used have been presented in section 5.2.

In this chapter, we assess the performance of a specific hardware that we have identified as a potential suitable candidate: the TILE-Gx. This hardware is benchmarked relative to a wide range of expected ELT instruments requirements and exhibits very good performance. We begin this chapter by discussing the specific characteristics of the TILE-Gx (6.1). We then present the test facility used (6.2) and detail the experimental results in 6.3 and 6.4.

6.1 TILE-Gx36

The TILE-Gx is a multi/many-core processor family produced by Mellanox. It consists of a mesh architecture of cores ranging from 9 up to 72 CPU cores. An illustration of the processor block diagram is given figure 6.1. These devices have been designed primarily for applications such as networking, video encoding or cloud computing. They are therefore designed mainly to provide many powerful CPUs and good power efficiency. This allows the Tile-Gx series to offer a flexibility of usage, especially when compared against special-purpose processors such as DSPs or FPGAs. One of the main features of the TILE-Gx

family cards is that they offer multiple 10 GbE I/O ports. Data can be rapidly and very efficient delivered to the cores for processing. In this chapter, we investigate the TILE-Gx36 which as with all COTS hardware is not designed specifically for use in a AO RTC. The TILE-Gx36 is a many-core processing card with 36 cores and four 10 GbE Ethernet ports. The TILE-Gx36 is part of the TILE-Gx range which offers multiple options for clock frequencies and other technologies such MiCA Accelerator for encryption. The specifications of the model we are testing are shown in table 6.1.

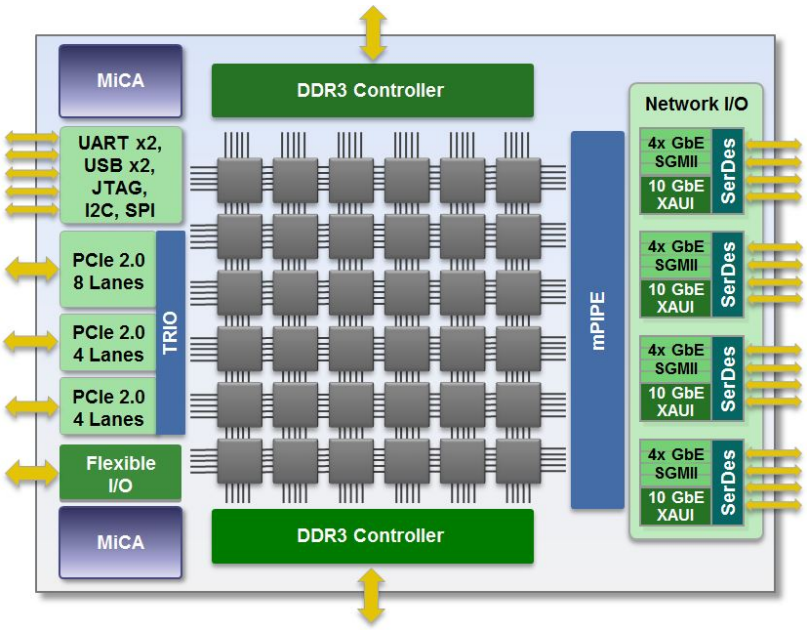


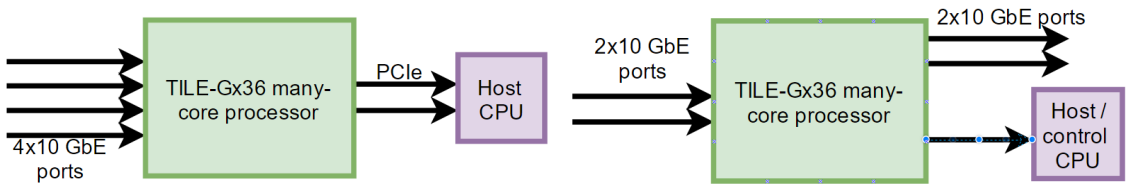
Figure 6.1: TILE-Gx36 Processor Block Diagram[1].

Table 6.1: TILE-Gx36 specifications.

Processor		Tile-Gx36
Device ID		TLR4-03680CG-12C-A3b
Number of Cores		36
Clock Frequency		1.2 GHz
cache	Total	12 MB
	L2	256 Kb per core
	L3	9 MB Coherent
PCIe		2.0
Memory		DDR3

The TILE-Gx comes with a limited set of specific libraries and a development environment called the 'Tilera Multi-core Development Environment' (MDE). The MDE offers compilers, an IDE and libraries for the TILE processors. This environment supports development in languages such as C/C++ as well as Java. The Tile-Gx processors run a micro-Linux kernel. Since the Linux mainline release of 2.6.36 in October 2010, there has been official support of the TILE architecture in the Linux mainline. Developing software for the TILE architecture is therefore very similar to developing conventional software for standard x86 instruction set CPUs.

The TILE-Gx series can fill many computation roles. Figure 6.2 shows two block diagrams of the TILE-Gx36 filling the role of a front end processor controlling I/O for a host computer. It also shows how the TILE-Gx can be used as a node in a dataflow pipeline. Using the TILE-Gx as a WPU, and depending on the actual architecture of the AO RTC system, either of these methods could be used. This chapter, while investigating the performance of the TILE-Gx hardware, is independent of the actual AO RTC system architecture choice.



(a) (b)

Figure 6.2: Block diagrams showing how the TILE-Gx36 could be used in (a) High I/O front end application and (b) In line as part of a dataflow pipeline.

The TILE-Gx family has mainly been designed to be used in data centres and to perform networking tasks such as packet filtering, bandwidth management, SSL/IPsec security protocol processing and high density video transcoding. While it is not designed to be the WPU for an AO RTC system, the TILE-Gx theoretically offer the computational power (i.e. with the many-core environment needed for ELT scale systems) as well as the high I/O capacity capable of coping with the high data throughput delivered by WFS.

Many other technologies can naturally be used for the WPU. In particular, the type of processing required is particularly well adapted to FPGAs. Many modern systems currently use FPGAs for this task; they are for example used in SPARTA on SPHERE. This is due to FPGAs being suitable to perform small amounts of processing on large amounts of data. The major downside is typically the development time associated with programming in low level languages such as VHDL or Verilog and the high initial implementation costs.

Modern tools are however helping bridge this gap. Tools such as OpenCL allows FPGAs programmed from C/C++ (see section 4.3.1.3). OpenCL is also available for co-processing cards such as GPUs and Xeon Phi and could offer a good alternative for platform independent devel-

opment. Several platforms exist to accelerate development time for non-FPGA experts. For example, Quickplay is software-centric design platform for FPGA developed by Accelize for non-FPGA experts, a spin out from PLDA. It offers a modeling tool allowing the user to connect a series of functional blocks. These blocks can either be developed in C/C++ by the user or for common functionality (i.e. encryption/compression) can be purchased from an in-built store. When development is complete it can then be deployed on any supported FPGA board.

These tools can potentially allow the reduction of development time and costs. However, they are also at the beginning of their life cycle and are subject to changes which may impact many developments, especially performance or upgradability. We have decided to use COTS hardware (Tile-Gx) which allows the development to be much closer to the standard x86 development done in C/C++. It allows greater flexibility for future upgrades and enables us to stay hardware independent for as long as possible to reduce future maintenance and development costs.

6.1.1 Memory Bandwidth

An important indicator of the achievable performance of computer processing cards, alongside the number of available cores and the clock frequency, is the memory bandwidth which can be extremely important for certain classes of computational tasks. The memory bandwidth is the rate at which data can be read or stored from memory by the processor. It can have a large impact on performance: if a core cannot access data in memory, calculations cannot be performed in time which leads to long latencies. Furthermore, when developing an application,

it is important to understand memory access issues in order to optimise for the available cache size and reduce the number of calls needed to slower memory.

In the WPU, large amounts of data are received from wavefront cameras, processed, and then output. If the memory bandwidth is too low, there will be a bottleneck in the data processing, leading to long delays and increased latencies.

To measure the memory bandwidth of the TILE-Gx, the STREAM memory benchmark[2] was used. STREAM is a benchmarking tool developed to assess the memory bandwidth of hardware using data sets many times larger than the cache of the chip. More information can be found in section 4.2.3.2, in particular about the different tests and outputs of the STREAM benchmark (i.e. Copy, Scale, Sum and Triad). This tool has been used by many groups and has produced a wide survey of the achievable memory bandwidths of a wide variety of computational hardware.

The STREAM memory benchmark performs four different routines to measure the memory bandwidth. The tests we are most interested in are the TRIAD, and to a lesser extent, the SUM. In the WPU the Centre of Gravity (CoG) algorithm (see section 5.2.3.1) for each pixel in a sub-aperture, there will be two TRIADs and a SUM performed. The attainable memory bandwidth will sit somewhere between the SUM and TRIAD results.

Table 6.2 shows the results of the STREAM memory benchmark. The results show that the TILE-Gx has a memory bandwidth of approximately 12 GB s^{-1}, this is low compared to other commercial off-the-shelf (COTS) technologies. For comparison a mid-range proces-

sor used in standard desktop computers (Xeon i5-4690) has a memory bandwidth of 25 GB s^{-1}[3] and the computer this thesis was written on (AMD Fx 6300) has a memory bandwidth of up to 29 GB s^{-1}.

Table 6.2: Tile-Gx36 STREAM results.

Function	Rate (MBs^{-1})	RMS time (s)	Min time (s)	Max time (s)
COPY	10985.1	0.097962	0.097745	0.101567
SCALE	12536.7	0.085992	0.085648	0.094419
SUM	12381.5	0.130223	0.130082	0.133454
TRIAD	12425.6	0.129882	0.129621	0.134147

6.1.2 Zero Overhead Linux

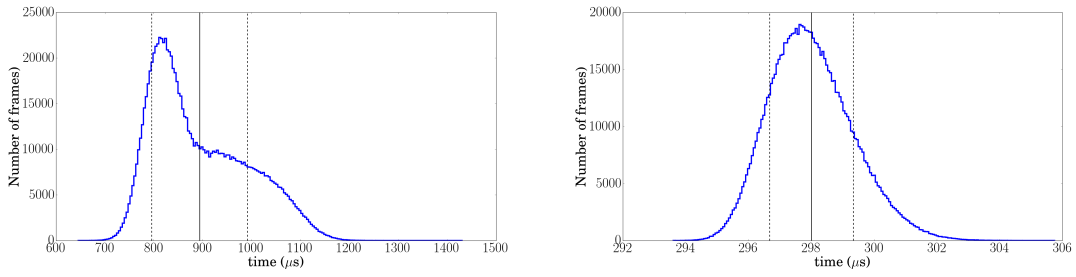
6.1.2.1 ZOL Mode

In normal operation, the TILE-Gx runs a non real-time micro Linux kernel, each core will be affected by standard Linux interrupts. One of the more interesting and useful modes of operation of the TILE-Gx is the Zero Overhead Linux (ZOL) mode. In the ZOL mode, the TILE-Gx offers the possibility for the user to specify a subset of tiles (i.e. cores), each of which will run a single, user-space task, without incurring any Linux system overheads[4]. The specified cores are free of all Linux system overheads and interrupts. The ZOL mode allows for near real-time performance and is capable of removing major variations in execution time. This is an important aspect for the WPU, to perform fast and stable pixel processing and is naturally the mode that has been chosen to test the TILE-Gx.

6.1.2.2 Comparison of non-ZOL and ZOL performance

Figure 6.3 presents histograms for the execution time of the TILE-Gx performing an image calibration followed by CoG calculations (see section 5.2.3.1)) using a 500×500 WFS detector. These calculations are repeated 10^6 times. The calculation is performed both using ZOL as well as in normal operation (i.e. non-ZOL). In normal operation the pthreads implementation is used while in ZOL mode threading is performed using the TILE-Gx specific libraries.

Not only do we see that the ZOL mode has a lower mean (298 μs vs 900 μs), but also that it reduces the variation in execution time referred to as jitter (calculated as the standard deviation).



(a) (b)

Figure 6.3: Comparison between non-ZOL and ZOL modes of operation and its effect on the wavefront processing time and stability. (a) Non-ZOL and (b) ZOL

Table 6.3 summarises the main results obtained in figure 6.3 and further illustrates how performance can be increased by using the ZOL mode. We can observe a strong reduction in mean execution time (a factor 3) and a reduction in standard deviation by a factor of almost 100.

Table 6.3: Summary of the main quantities obtained in figure 6.3 comparing calculation times using ZOL and non-ZOL modes. Values are given in μs.

Mode	mean	σ	min	max	range
Non ZOL	893.66	98.00	645.34	1430.21	784.86
ZOL	298.72	1.33	292.72	304.252	12.151

For this test we used a detector of size 500×500 pixels, not all WFS detectors for future AO systems will be this size and it is important to understand how ZOL and non-ZOL modes evolve with detector size. Figure 6.4 shows the image calibration and CoG calculations performed by the WPU for both ZOL (blue) and non-ZOL (red) modes. Each time measurement is repeated 10^6 times per detector size. For each data point, the plot also shows the standard deviation (black) and the total measured range (green).

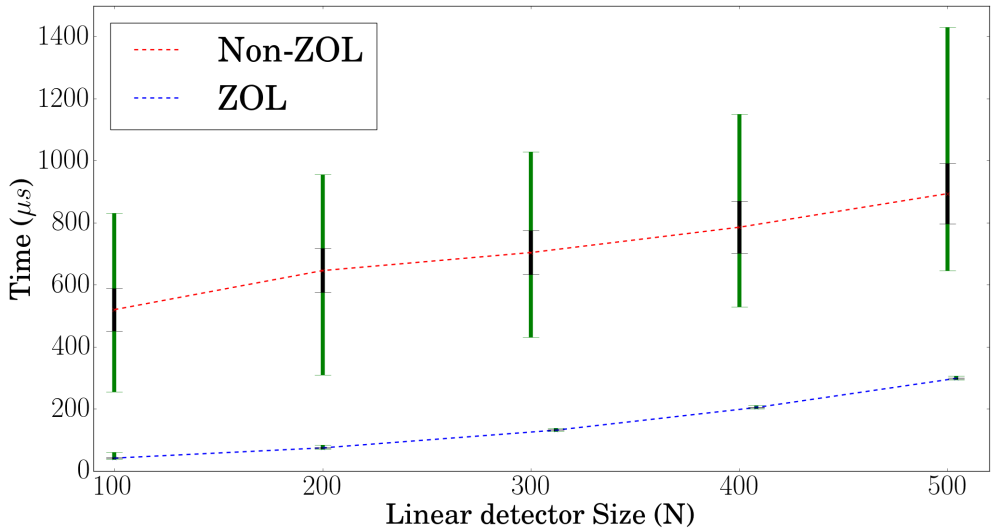


Figure 6.4: Comparison between non-ZOL and ZOL modes of operation for the scaling of the detector size (the full detector size being N×N). Red is non-ZOL results and blue is ZOL results. The vertical bars represent the standard deviation (black lines) and the total measured range (green lines).

Table 6.4 summarises the main results obtained in figure 6.4 and

further illustrates how performance can be increased by using the ZOL mode across multiple problem sizes. We can observe a strong reduction in mean execution time and a reduction in standard deviation, which is more pronounced for large system sizes. The variation in execution time for the ZOL mode are very stable, staying at the level of $1.3\mu s$ regardless of the detector size. The ZOL mode very convincingly offers both increased mean performance and increased stability (determinism).

Table 6.4: Summary of the main quantities obtained in figure 6.4 comparing calculation times using ZOL and non-ZOL modes. Values are given in μs.

size	ZOL			non-ZOL		
(N×N)	mean	std	range	mean	std	range
100	41.163	1.312	22.685	519.61	68.338	576.048
200	74.152	1.335	13.9	645.551	70.769	645.809
300	131.252	1.323	11.743	703.694	70.465	597.897
400	204.158	1.268	12.008	785.256	84.533	620.083
500	298.007	1.329	12.151	893.657	98.006	784.862

The significant reduction of the mean and jitter in ZOL mode can come from multiple sources. Using the ZOL mode, we reduce the operating system (OS) overheads and stop the OS from interrupting calculations. We believe the major improvements in performance actually come from using the hardware specific libraries and not only from using the ZOL mode. We make this assumption due to the fact that the effect of moving from the standard libraries to the ZOL and proprietary libraries shifts of the entire distribution. If this was an effect of only removing the OS activity and interrupts, we would likely see a reduction in the outliers but not the entire distribution.

It is unlikely that the impact of the OS would be seen at every iteration, in other words the impact of the OS is periodic and not a

constant. A more likely scenario is that the OS impacts only a subset of iterations and therefore creates outliers. As we see a shift in the entire distribution and not only the outliers, we believe that the improved performance is due to a better optimised algorithm provide by a different implementation of the desired calculation.

The hardware specific libraries are optimised specifically for the TILE-Gx and provide the main performance improvements. Since it is not possible to use standard C/C++ libraries in the ZOL mode, proprietary libraries must be used. This could potentially increase development time but ensures that we achieve the very best possible performance. For the rest of the chapter we will use both the vendor-supplied hardware specific libraries and the ZOL mode, unless otherwise mentioned.

6.1.2.3 ZOL mode limitations

When using ZOL mode, the TILE-Gx does not allow certain functions to be used. These functions typically are system level functions such as printing to screen (`printf` and `cout`) or debugging methods. This can lead to some difficulty debugging code. For this reason the TILE-Gx comes with the MDE to help the development and reduce time to market of products.

Shared memory

The TILE-Gx libraries include a proprietary implementation of a multi-threading library. It has been developed to optimise the TILE-Gx available resources and it is recommended over the standard C/C++ libraries. This library uses very similar interfaces to pthreads and allows for simple portability of code as shown in listing 6.1. This is a exam-

ple of the Tile-Gx specific code when compared against pthreads the standard multi-thread library available in C. This shows the similarities between the libraries interfaces and how code can be developed to be portable without making any major changes to the code structure.

Listing 6.1: Code example in C illustrating the portability of software specifically written for the TILE-Gx to another platform using standard multi-threading libraries.

```
#ifdef TILEGX
   pthreads_mutex_lock(pthread_mutex_t mutex);
#else
   tmc_spin_mutex_spin_lock(tmc_spin_mutex_t mutex);
#endif
```

Another important aspect of the ZOL mode is that each thread is a single user space task with no access to the global memory. For example, using malloc to allocate memory will work within a single thread, but this memory cannot be accessed by other threads. To allow threads to access and share memory, the memory has to be allocated using the TILE-Gx libraries and has to be flagged as shared memory; this is shown in listing 6.2.

Listing 6.2: Example of C code to allocate memory that can be accessed and shared by all threads.

```
tmc_alloc_t alloc;
tmc_alloc_set_shared(&alloc);
int* data = tmc_alloc_map(&alloc,
                          length*sizeof(int));
//do something

// free memory
```

```
tmc_alloc_unmap(data, length*sizeof(int));
```

Debuggers

The profiler, for example, gives information on bottlenecks or inefficiencies in the synchronization between threads and helps the developer to optimise complex multi-threaded code. Both of theses tools typically interact with the OS and for this reason cannot be used in ZOL mode. This can add complexity during code development since other standard debugging techniques (such as print variable to screen) are also not available. The standard debugger was used during code development but was not included during the actual TILE-Gx performance tests.

Timings

It is crucial in our application to be able to measure the execution time of sections of code accurately. In standard C/C++, the profiler is typically used to measure times. However, it adds unwanted overheads that influence the overall timing accuracy. Another possibility is to use the Linux clock. In that case, we measure the difference between the time before and after the section of code of interest. This method has been shown to be accurate but unfortunately doesn't work in ZOL mode because of the need to access system levels functions. The MDE also offers multiple tools to measure performance in ZOL mode. There are multiple techniques offered though they all stem from clock cycles between two points in the code.

Because of the above limitations, we have decided to measure performance using the simplest instance of counting clock cycles. This

method is illustrated in code listing 6.3. This method requires the specified clock frequency to be accurately known to give an absolute time measurement. We believe that this limitation is in reality negligible, especially when comparing relative performance. We have compared this method to using the Linux clock in non-ZOL mode and have found them to give identical results.

Listing 6.3: Code example to calculate execution time of a section of code based on the CPU clock frequency.

```
start_count = get_cycle_count();
// Do stuff
some_function()
end_count = get_cycle_count;
clock_frequency = tmc_perf_get_cpu_speed();
time = (end_count - start_count)/(clock_frequency);
```

6.1.3 TILE-Gx I/O: MPIPE

One of the main functions of the WPU is to receive wavefront camera images at a high frame rate. The TILE-Gx Ethernet ports are required to receive large amounts of data at high speed without losing any packets. Sockets are the standard technique to send data over Ethernet when working with Linux in C/C++. The standard socket libraries offer a standardised way of processing I/O Ethernet data and involve high levels of interaction with the OS.

standard library sockets are not just implementations in the programming language (i.e. C/C++), they are constructs in Linux and allow an interface for applications to interact with the hardware. There are many different levels of interactions that the developer typically

does not need to have access to. These can be broken down into:

- Application level
- Transport layer (UDP/TCP)
- Internet layer (IP)
- Network interface card, (NIC) (Ethernet, WIFI, etc.)

The only layer the developer interacts with is typically the application layer, the rest is handled by the OS. This leads to difficulties when optimising systems as the typical received packet by a socket is handled as follows,

- Packet received at hardware NIC.
  - An interrupt is generated.
- OS interrupt handler moves data from hardware buffer into a main memory
  - Packet is placed in a queue[1]
- The OS processes the packet in the protocol stack for IP UDP/TCP processing.
  - Data is sent to the corresponding socket (e.g. IP address and port number).
- Packet is queued in socket queue.
  - Checks if any application needs packet. If needed, the packet is copied from queue to user defined buffer.

[1]In Linux only a single receive queue exists for all network interfaces

The queues are typically ring buffers, which are an allocation in memory where data can be stored. Once the buffer has been filled, it will start overwriting the previously stored data at the beginning of the ring. Ring buffers gives a small memory allocation for an infinite amount of incoming data. They are particularly useful in scenarios with data streams or when data in temporally stored while being processed. Due to the limited storage capability, ring buffers are problematic if data is required to be permanently stored in memory.

Unfortunately the standard library sockets are unusable in ZOL mode as they make system level calls from the hardware interrupts supplied by the Network Interface Card (NIC) and interact with the OS. The TILE-Gx comes with its own libraries to handle I/O called Multicore Programmable Intelligent Packet Engine (MPIPE). MPIPE offers low level API to control the data communication. This allows complete control over the queue buffers and registers to control the packet classification, load balancing, and buffer management. This requires much more developer involved initialisation than standard sockets, but once the MPIPE engine has been initialised its use is not too dissimilar to that of sockets. In addition, and as shown in section 6.1.2, the standard libraries do not appear to perform as well as the hardware specific libraries that are supplied by Mellanox. Although using MPIPE libraries increases development time and reduces code portability, they offer far greater performance.

Since the OS cannot be accessed by cores in ZOL mode, the NIC has to be handled directly by the application rather than the OS. The application will need to initialise the NIC, allocate memory for the ring buffers and NIC registers, and decided the rules in which incoming data

will be handled.

To understand how the TILE-Gx handles the I/O of data using both sockets and MPIPE we have streamed images to the TILE-Gx over 10 GbE. Table 6.5 shows the number of Ethernet packets that the TILE-Gx has received by the TILE-Gx card after sending 10^6 image frames. Each frame contains 640×640 pixels with 8 bit pixel depth, giving each frame ≈ 0.4 MB of data or 285 standard size (1500 Bytes) Ethernet packets. It was not possible to use jumbo frames (9000 Bytes) since the Linux kernel version used on the TILE-Gx did not support them at the time of the tests[2].

Data is streamed at the fastest rate the FPGA Pixel Emulator (see section 6.2) is able to; which for this example is 7.5 Gbit s^{-1}. If we assume a first light E-ELT instrument with a single 800×800 WFS and a frame rate of 1 kHz the WPU would need to be receiving data at a rate of ≈ 5 Gbit s^{-1}[5]. Here we are testing rates slightly higher than may be used in a first light instrument but will give a good indication of the TILE-Gx performance at high data rates.

The TILE-Gx is tested under three different modes of operations; the first being standard library sockets. The second two use the MPIPE API with and without using ZOL. From table 6.5 it is obvious that the same problems we saw in section 6.1.2 are also present here. There are no missed packets when using MPIPE in both modes, while the socket implementation only managed to receive ≈ 25% of the data packets. Considering the poor performance of the standard C/C++ libraries relative to the hardware specific libraries (see section 6.1.2), it is not surprising to see that the MPIPE implementation can easily handle the ingress of data whereas the socket implementation struggles.

[2]A new kernel version has been since released that supports jumbo frames on the TILE-Gx.

Table 6.5: Frames received by the TILE-Gx in different modes of operations

Method	packets received	packets missed	percentage received
Sockets	69554011	215445989	24.4%
MPIPE	285000000	0	100%
MPIPE ZOL	285000000	0	100%

Figure 6.5 shows the histograms of the ingress of data onto the TILE-Gx for each image frame. Table 6.6 summarises the main values from the distributions shown in figure 6.5. The socket implementation shows much worse performance compared to either the MPIPE or MPIPE ZOL implementation and does not have an acceptable level of performance for real-time system applications.

For the MPIPE we see similar results from the ZOL and non-ZOL implementations. The difference in distribution is minor with a slight reduction in overall range. From this alone either implementation should suffice. A maximum of two cores were under load during these tests, leaving 34 idle cores where the OS background tasks are able to be run and not impact the results. As the load on the TILE-Gx increases it is increasingly likely that the non-ZOL variance would be increased as scheduling issues and system overhead increase. This could lead to some OS tasks being performed on cores running WPU computations. Since the ZOL mode removes these issues from specified cores the ZOL distribution would remain unaffected, unless the system was running at the memory bandwidth limit.

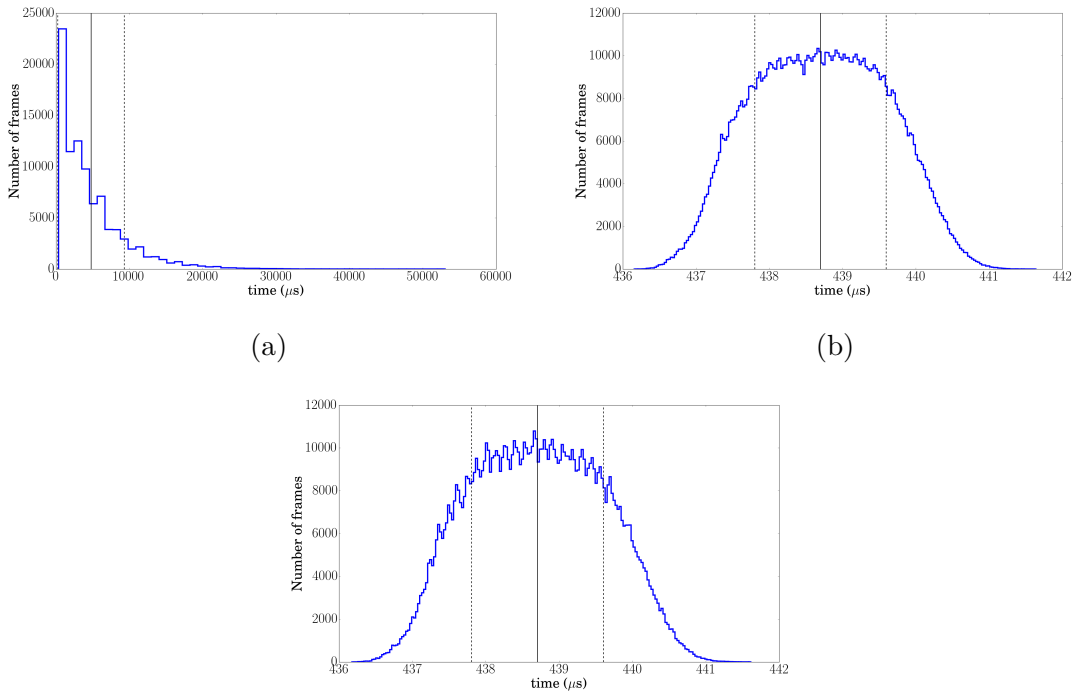


Figure 6.5: Comparison between standard library sockets (a) and the MPIPE in ZOL (b) and non-ZOL modes (c) of operation and the effect this has on the receiving Ethernet packets.

Table 6.6: Frames received by the TILE-Gx in different modes of operations

Method	mean (μs)	std (μs)	range (μs)
Standard library sockets	4760.27	4523.33	52690.60
MPIPE	438.70	0.897	5.48
MPIPE ZOL	438.70	0.897	5.43

From these data sets it is clear that when handling the I/O of data on the TILE-Gx, the MPIPE API is needed. This allows the TILE-Gx to achieve the best I/O performance, when high data rates are used. Throughout the rest of this chapter, where I/O of data is performed with the TILE-Gx the MPIPE will be used.

6.1.4 Thread affinity and priority

When building multi-threaded applications on many-core processors it is important for the developer to specify the core that the processes will run on to make sure that any real-time processes are run on a single core. Running on a single core allows the maximum performance and avoids context switching.

The TILE-Gx offers tools for setting the affinity of threads, though in a different way to how this is typically done on Linux. Specifying the tiles for the process to run on is especially important for ZOL mode as this interferes with the standard Linux scheduler.

It is important for time critical applications running on an OS to have raised priorities. The scheduler in the OS decides what process to run when and where based on that priority. It can make this decision using different metrics and also depends on the scheduler mode, more information on schedulers in section 4.3.3. One of the main factors in this is the priority of the process. Processes with higher priority are typically performed before those with lower priority.

When the TILE-Gx is in ZOL mode the OS is not interrupting on specified tiles, as long as only a single thread is allowed on each tile so no context switching can takes place, priority being raised will make no difference.

6.1.5 Impact of system parameters on performance

6.1.5.1 Impact of number of cores on performance

The next generation of TILE-Gx, the TILE-Mx series was announced to be released in 2016[6]. After the acquisition of EZChip by Mellanox, there have been no subsequent announcements of the new chips. It is

expected that the new TILE-Mx will use ARM processors and expand the number cores up to 100. The chips are also expected to use DDR4 RAM (instead of DDR3 currently) giving the TILE-Mx access to higher memory bandwidths.

With such a wide range of possible number of cores, it is important to understand how this number impacts performance. We chose to perform the full-frame wavefront processing (see section 6.3 for more information on full-frame) where we wait for a complete WFS image frame to be received before proceeding the data.

Figure 6.6 shows how the mean performance of the TILE-Gx36 scales with the number of cores being used for processing. Table 6.7 enumerates a selection of cores corresponding to current and announced TILE-Gx series. From the measured data, we are able to fit a $\frac{1}{\text{number of cores}}$ to extrapolate beyond 36 cores. This fit function is described Amdahl's law (see section 4.2.2) to characterise the performance increase when using multiple cores for a parallelisable and fixed problem size.

The wavefront processing calculations performed on the TILE-Gx are shown to scale predictably with number of cores being used. For example, we would be able to decrease the mean processing time by nearly a factor of 2 (i.e. 1.97) by simply increasing the number of cores from 36 to 72. And by increasing from 16 to 64 cores ($\times 4$ cores) we see a decrease in the mean processing time by a factor of 3.9.

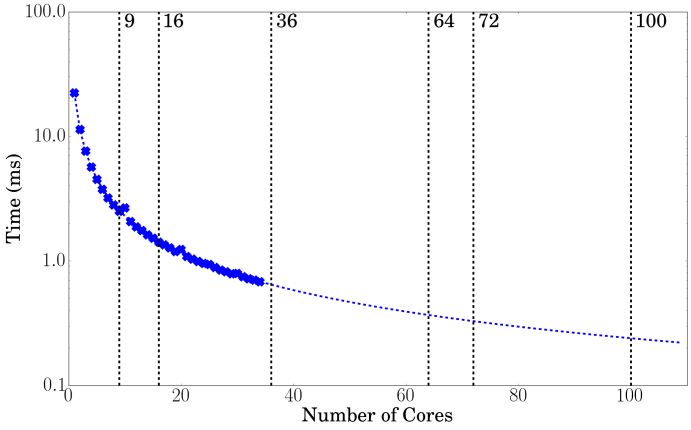


Figure 6.6: Mean processing time t_{wp} as a function of number of cores used. A $\frac{1}{\text{number of cores}}$ fit has been added to enable a performance prediction for larger systems. Dashed vertical lines represent current and expected number of cores.

Table 6.7: Mean processing time (values extracted from figure 6.6).

Cores	time (μs)
9	2546.4
16	1433.1
36	643.4
64	366.8
72	327.3
100	238.7

6.1.5.2 Impact of number of clock frequency on performance

Clock frequencies for computational hardware has been increasing and continuing to increase. To estimate the performance of how this increase in clock frequency will impact the performance of the TILE-Gx we have calculated the performance with a scaling clock frequency in figure 6.7. We have calculated this under the assumption that doubling the clock frequency will half the computation time. This is likely to hold true as long as the memory bandwidth is not limiting the calculation.

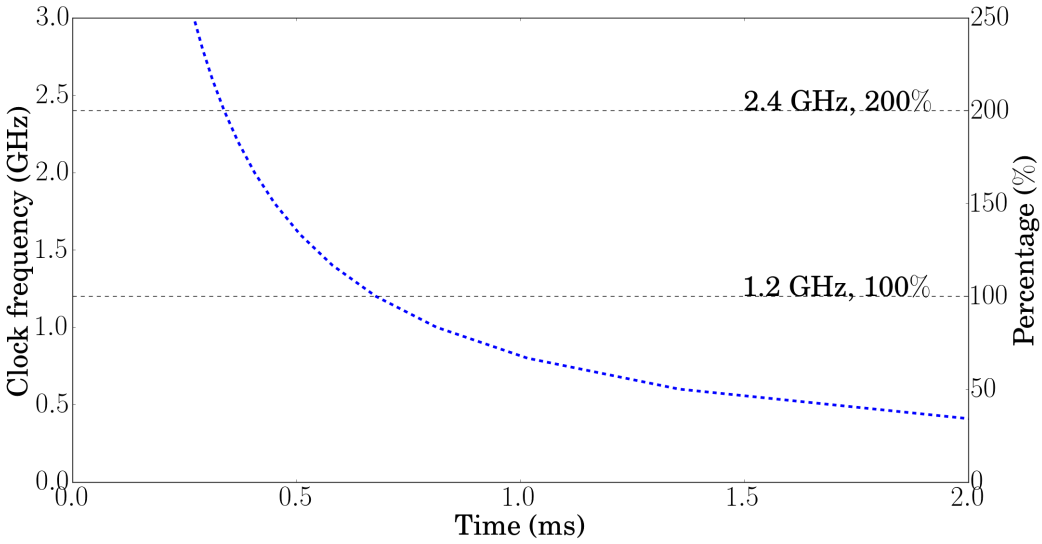


Figure 6.7: Mean processing time t_{wp} as a function of clock frequency.

6.1.5.3 Impact of memory bandwidth on performance

In section 6.1.1, we investigated the memory bandwidth of the TILE-Gx and found it to be relativly low at ≈ 12 GB s^{-1}. While the computation being performed is not inherently memory bandwidth limited like some BLAS functions, the low memory bandwidth may limit the calculation time if the number of cores or clock frequency are increased in the future. The impact of increasing the memory bandwidth on the TILE-Gx is hard to quantify as the calculations are not memory bandwidth limited. In addition, if the number of cores or clock frequency is increased, the memory bandwidth would also need a increase to make the most out of the increased resources. Increasing the memory bandwidth is unlikely to increase the performance for applications using MPIPE libraries. For standard library tasks it could make a difference.

6.2 Testing facility

To fully test the capabilities of the TILE-Gx, the ability to stream data over 10 GbE is essential. We worked with the Rutherford Appleton Laboratories to develop an FPGA card with the ability to stream data over 10 GbE at chosen rates. This lead to the development, by the Rutherford Appleton Laboratories, of what we are calling the Pixel Emulator, which is based on a pixel emulator developed for the European X-ray free electron laser[7]. We can upload a series of images to the FPGA board memory of chosen size, that can then be sent over 10 GbE Ethernet at a chosen data rate. We are able to specify the number of bits per pixel, in this chapter we will be using 8 bits per pixel. A picture of the Pixel Emulator is presented figure 6.9 and the overall architecture used for testing the TILE-Gx36 is shown figure 6.8.

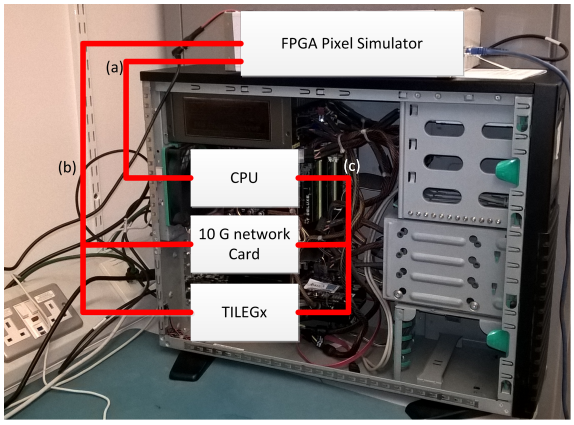


Figure 6.8: Shows the FPGA pixel Emulator and how it is connected to the PC and TILE-Gx36. (a) a 1 Gbps Ethernet link that allows for controlling the FPGA Pixel Emulator. (b) 10 Gbps Ethernet link that is used to stream the pixels from the FPGA Pixel Emulator and allows uploading images to the FPGA Pixel Emulator. (c) the PCIe bus link that connects the TILE-Gx to the computer.

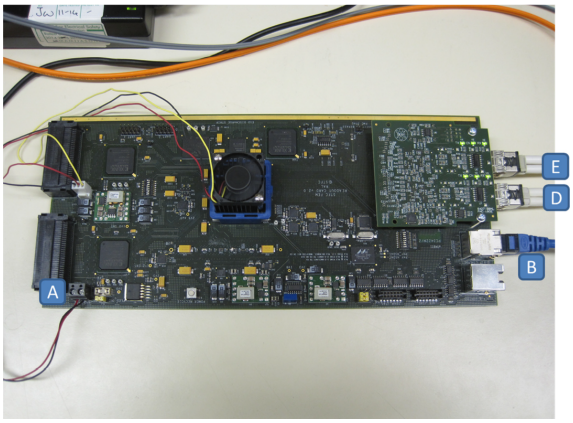


Figure 6.9: Shows the FPGA pixel Emulator board and connectors. (A) Power supply. (B) a 1 Gbps Ethernet link that allows for controlling the FPGA Pixel Emulator. (D) 10 Gbps Ethernet link that is used to stream the pixels from the FPGA Pixel Emulator and allows uploading images to the FPGA Pixel Emulator. (E) 10 Gbps Ethernet link that is used to upload data (images) to the FPGA Pixel Emulator.

Using the FPGA Pixel Emulator, we are able to stream known wavefront images at known data rates to a WPU, in our case this is the TILE-Gx. The pixel emulator gives us great flexibility over using an actual camera to stream pixels. It can stream any image size, at any given data rate and with different numbers of bits per pixel, making it extremely using to rapidly test a wide range of potential WPU systems. To demonstrate the ability of the FPGA Pixel Emulator we have streamed 10^6 frames of size 640×640 (0.41 MB) pixels to the TILE-GX where no processing is taking place. The results of this are presented in Figure 6.10 and Table 6.8.

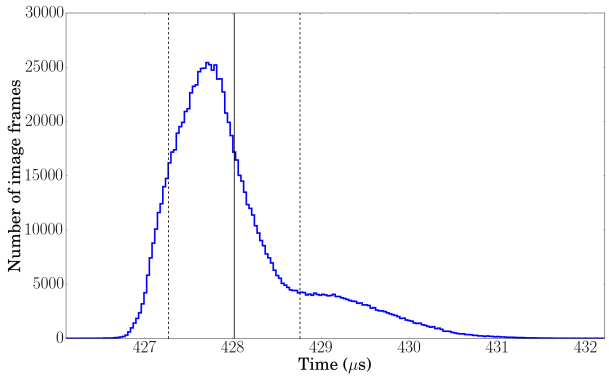


Figure 6.10: Variation in receiving data from FPGA Pixel Emulator for a 640×640 detector. Distribution calculated for a 10^6 frames. The black vertical line represents the mean and the dashed vertical lines the standard deviation.

Table 6.8: Time variation in receiving data from the FPGA pixel emulator (values extracted from figure 6.10)

Detector	640×640 pixels
Mean	428.01 μs
Standard Deviation	0.74 μs
Range	6.11 μs

In this example, the TILE-Gx did not perform any processing on the received data packets. The FPGA can output at rates of 7.5 Gbit s^{-1}, close to the expected rate from 10 Gbps Ethernet.[8]. The FPGA is actually designed to stream larger frames: if we increase the frame size from 0.41 MB to closer to the 16 MB buffer limit we would likely be able to get closer to 10 Gbit s^{-1}.

We see that both the FPGA and the TILE-Gx are able to maintain these data rates over long periods of time. No data packets have been dropped for the 1 million image frames sent, which equates to over quarter billion data packets sent. The histogram shows a small range and a sub-microsecond standard deviation. It clearly demonstrates the capacity of the both the FPGA and the TILE-Gx to cope with large amounts of data at high data-rates without introducing noticeable jitter or latency relative to the overall AO real-time operation. In the rest of the chapter, we will therefore neglect any effects (i.e. jitter) caused by data transfer between the FPGA and the WPU.

In the future we hope to develop an external trigger. The 2nd RJ45 connector has 3 LVDS inputs which could be used for trigger signals. This would allow the FPGA Pixel Emulator to properly emulate a real wavefront camera and would allow the FPGA pixel Emulator to be integrated more easily into AO RTCs during development and testing. This could potentially be used during the commissioning of new instruments, replacing during this phase the actual wavefront sensing cameras.

6.3 Full-frame testing

6.3.1 Experimental set-up

We begin our investigation with full-frame testing where the TILE-Gx waits for a whole frame to arrive before processing the data. A timing diagram of this method is shown in figure 6.11. In order to disentangle the different sources of time delay and actual limitations of a real camera, we have decided to store a known wavefront image directly into memory, so no data transfer into the TILE-Gx occurs. This mode allows quick testing of many different size detectors, algorithms and the ability to accurately measure performance. In some particular cases, the RTC computation is actually delayed until the WFS read-out is complete. This method of operation could for example be used to WPU used for PYR-WFS sensors as well as infrared detectors subject to Fowler sampling read-out. In fact, in the case of a pyramid WFS, wavefront reconstruction cannot start until at least one pixel of the four sub-pupils have been read (specific detector read-out technologies can add additional constrains). While pyramid sensors benefit from being able to use smaller detectors, the WPU has to wait for longer before

being able to start processing the data. For WFSs with Fowler sampling read-out embedded in the camera, no data is provided to the RTC until correlation takes place and the final, pre-processed WFS data frame is output. No realistic parallelisation could take place here.

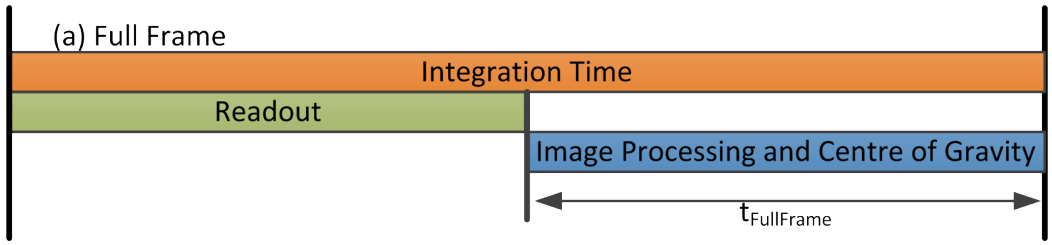


Figure 6.11: A simplified timing diagram showing the full-frame testing.

The sequence diagram of the code can be seen in figure 6.12. This diagram shows how full-frame tests are simulated, with no data being transmitted to the TILE-Gx. The main thread is the only thread not running in ZOL mode. The TILE-Gx in fact, only needs at least one core (tile) to run the OS in order to operate. The control thread does no processing but supervises the worker threads of which there are 34[3]. Only a single worker thread is represented in figure 6.12, as all worker threads are identical and representing 34 would make the diagram overly complex.

A timing is taken on the control thread when a broadcast message is sent to all worker threads saying there is work to be done. Each worker thread has an assigned list of sub-apertures that it will process; once it has processed them it will signal the control thread. The actual operation of each worker thread is discussed in section 5.2 and shown in figure 5.3. For each worker thread, the process is split in two separate steps, a calibration step (section 5.2.2) and a processing step (section 5.2.3). The calibration step is typically background subtraction, flat

[3]34 workers, 1 control and 1 running the OS totaling 36 threads for 36 tiles.

fielding and dark map subtraction. The processing step calculates the local wavefront slope of the sub-aperture under consideration.

Once all worker threads have signaled the control thread, another timing is taken. This workflow is then repeated. The worker threads will then wait until they receive another signal telling them to process data again. This process in repeated until a sufficient number of frames (typically 10^6) has been processed.

Timings are saved in memory while the program is running. It is only when the entire test is finished that timings are written to disk. This is done to minimise unnecessary processes running during the operations and to be able to measure timings accurately. This is also done due to ZOL mode making interactions with storage files more complex as it typically needs to interact with the OS.

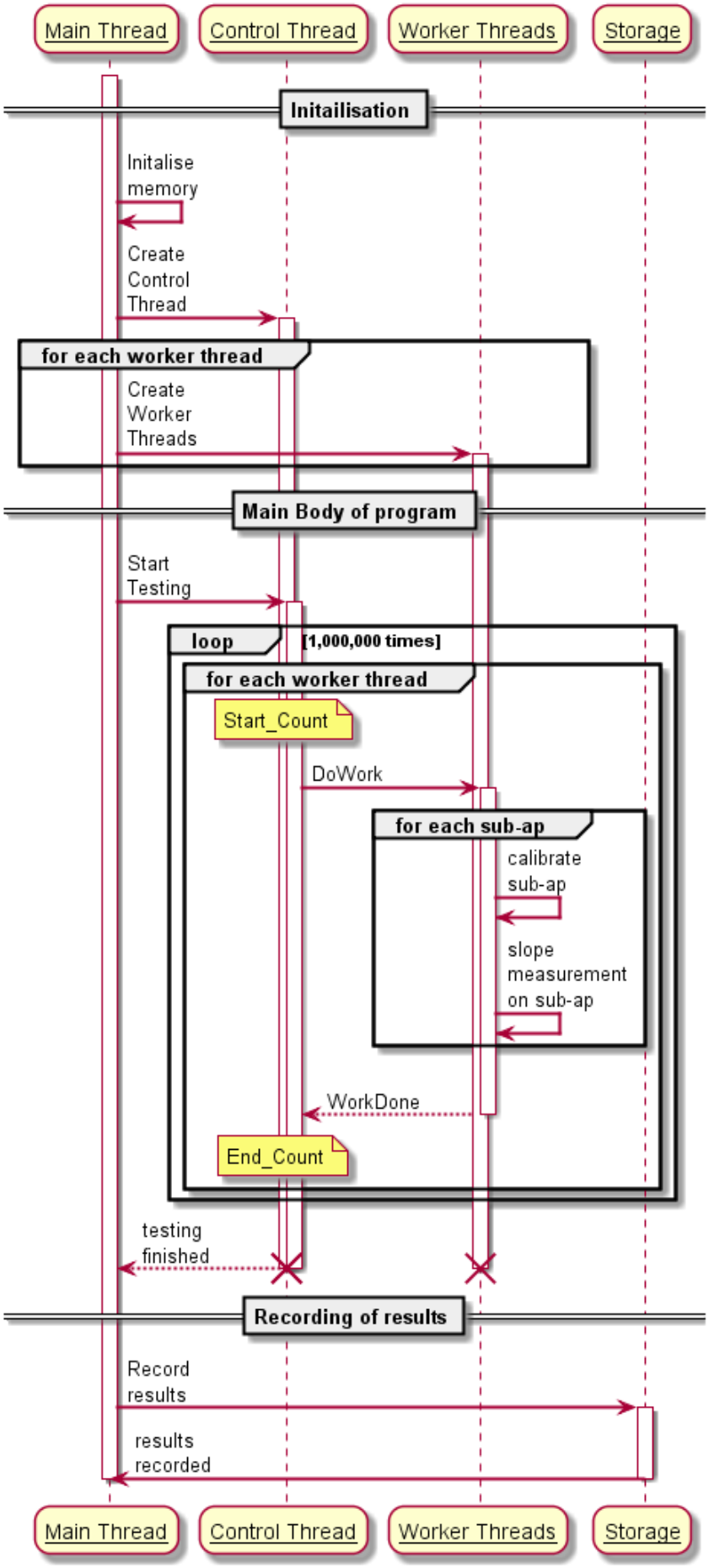


Figure 6.12: A simplified sequence diagram of the full-frame testing.

6.3.2 Impact of detector size on mean execution time

Figure 6.13 shows the mean calculation time (i.e. time from the last pixel arriving in the WPU to the last CoG being calculated) as a function of the linear detector size N (the full detector size being N×N). Three algorithms are tested and have been presented in section 5.2, namely CoG, weighted Centre of gravity (WCoG) and matched filter. The CoG (blue) is the simplest method of extracting the spot location from the wavefront image and performs the best out of each of the tested algorithms. The matched filter (green) and WCoG (red) take longer to complete than the CoG due to the added complexity of the calculation.

It is important to note however, that the mean computation time is very similar for all 3 algorithms and that they follow the same scaling law (i.e. N^2) as a function of the detector size. This gives us the confidence that implementing similar algorithms would not dramatically change performance. It also shows that presenting results in terms of one of the studied algorithms only will not restrict the conclusions one can draw from the experimental data.

We tested these algorithms for many different WFS detector sizes. The number of sub-apertures and number of pixels per sub-aperture have been varied, each data point (dots) representing a different combination of number of sub-apertures and number of pixels per sub-apertures. The execution time is clearly independent of how the detector is divided up in terms of total number of sub-apertures and size of the sub-apertures and is only dependent on the overall size of the detector.

Using this data, one can extract (and extrapolate) the wavefront

Table 6.9: Summary of the main quantities obtained in figure 6.13. Comparison of different WFS algorithms. Values given in μs.

Size	CoG	WCoG	MF
240	74.15	96.65	92.86
800	791.90	907.58	833.58
1200	1682.03	1922.07	1787.68
1600	2977.21	3322.69	3118.19

processing time of any given detector size. Table 6.9 gives a summary of the main considered WFS detector size for future E-ELT instruments.

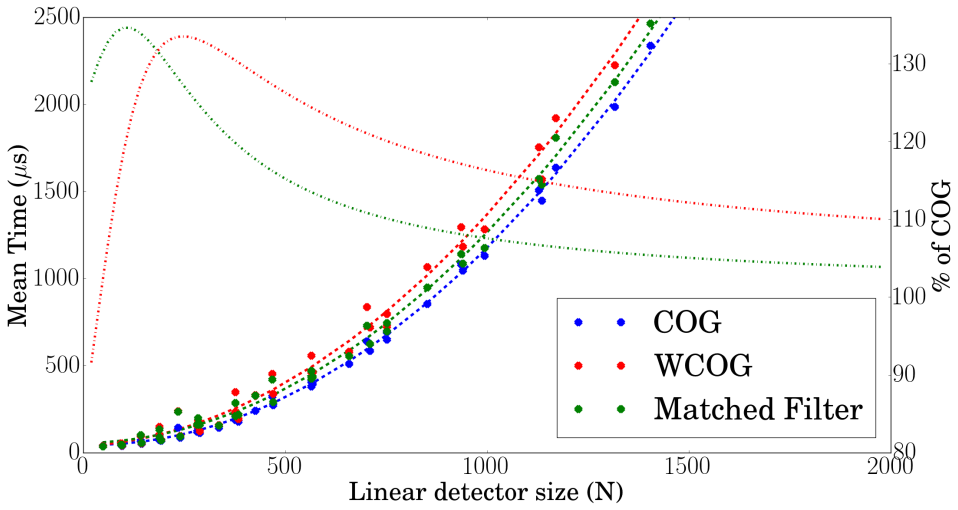


Figure 6.13: Mean WFS processing time as a function of linear detector size N for different algorithms (the full detector size being N×N). Dots: blue is CoG, red is WCoG and green is a matched filter. Each dot represent a different combination of number of sub-apertures and number of pixels per sub-apertures. The dotted curves is a N^2 fit. The series of dotted dash lines represent the percentage relative to the CoG calculation time.

6.3.3 Stability of the execution time

The mean performance is not the only important factor for an RTC. The variation of this execution time (or jitter) is also important. We show the overall distribution of execution times for a selection of detector

sizes in figure 6.14. We have tested these detector sizes for a million (10^6) frames. Three different detector sizes are shown, 200 × 200 (figure 6.14a), 500 × 500 (figure 6.14b) and 800 × 800 (figure 6.14c). The important values from the distribution are given in table 6.10. For each of these detector sizes we see that the system is very stable, with a distribution close to that of a Gaussian for the smaller detector sizes. To illustrate this a Gaussian has been plotted in green, using the mean and standard deviation from the measured data.

For the 800 × 800 (figure 6.14c) detector we see a double peak distribution that isn't seen in the smaller detectors. For this size the range is still low ($< 20\mu s$) and the standard deviation has slightly increased to $\sigma = 2.66$ μs.

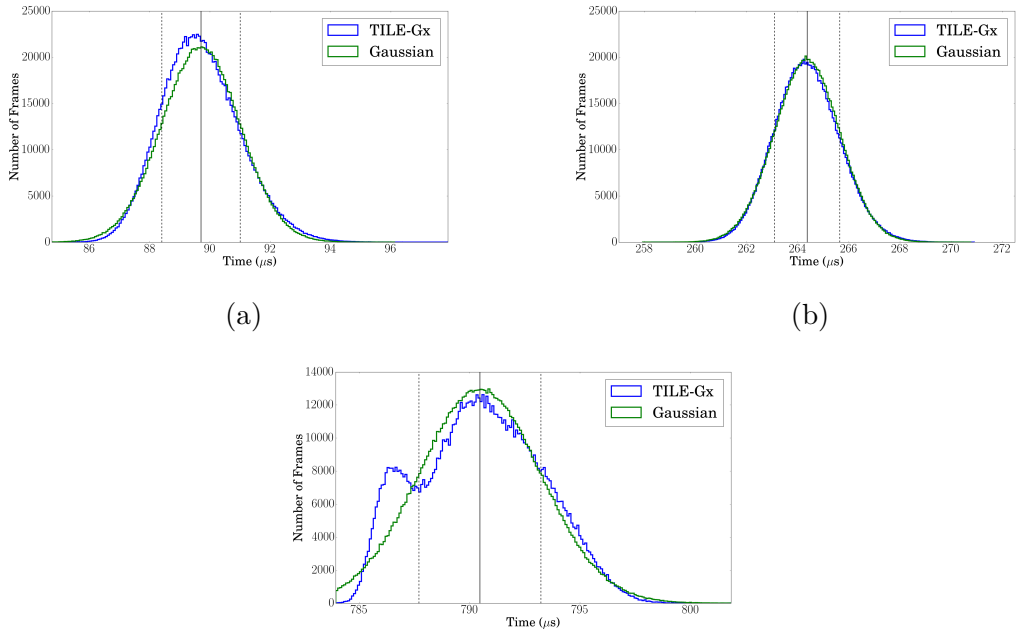


Figure 6.14: Variation in execution time for a selection of detector sizes using the CoG algorithm in full-frame mode. Distribution calculated for a 10^6 frames. The black vertical line represents the mean and the dashed vertical lines the standard deviation. Figure 6.14a 200 × 200, figure 6.14b 500 × 500, figure 6.14c 800 × 800

Table 6.10: Values extracted from figure 6.14

Detector size	mean (μs)	standard deviation (μs)	range (μs)
200 × 200	74.15	1.28	13.90
500 × 500	293	1.32	20.06
800 × 800	791.90	2.76	18.99

To better understand how the variation is affected by detector size, on figure 6.15, which shows the CoG from figure 6.13, we have over-ploted error bars to each data point to represent the standard deviation (black) and range (green). This plot clearly demonstrates that jitter (i.e. standard deviation) and range are extremely small compared to the mean value (they are barely visible on figure 6.15).

Due to these error bars to being too small to read we have replotted this data as a percentage of the mean the standard deviation (black) and range (green). Here we can see that for small systems the range is a large percentage of the overall time (> 20 %). This falls quickly as the range does not increase much in comparison to the mean time (< 5 %). Similarly, the standard deviation (σ) increases slower than the mean, falling from 1.7 % (200×200) of the overall processing time to 0.44 % (500×500) and 0.35 % (800×800).

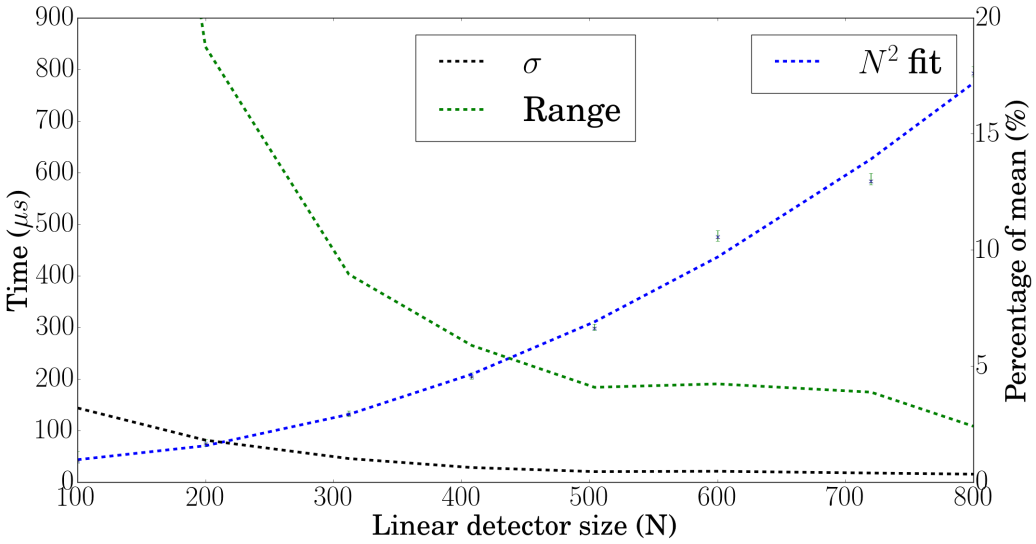


Figure 6.15: Mean WFS processing time as a function of linear detector size N for different algorithms (the full detector size being N×N). The dotted blue curve is a N^2 fit. The error bars shown to illustrate the standard deviation (black) and the range (green). The green and black curves represent the range and standard deviation respectively, as percentage of the mean.

If we know the data rate of a given wavefront camera we can estimate the time for a single frame to be read out from the camera to the WPU. Table 6.11 shows the values for the predicted camera read-out as well as the measure wavefront processing latency, if we assume the read-out rate for a camera to be 6 Gbps, which is slightly faster than the requirement for an E-ELT first light instrument[5]. For each of the shown detector sizes the wavefront processing time is comparable with the detector read-out time. The read-out time is typically less or equal to the integration time of the wavefront camera to minimise the latency of the AO RTC system.

Table 6.11: Latency caused by the camera read-out and wavefront processing unit for varying detector sizes.

Detector size (N)	read-out (μs)	wavefront processing (μs)	total (μs)
200	53.33	74.15	127.49
500	338.698	298.01	636.70
800	853.33	791.90	1645.23

6.4 Pipeline Testing

6.4.1 Experimental set-up

6.4.1.1 Timing definitions

The full-frame mode, where the system needs to wait for all pixels to be received before any processing is started, is far from optimal for SH-WFS (but might be the method of choice for pyramid WFSs or when using some infrared detectors). A more efficient configuration, as depicted in figure 6.16, can be obtain by starting the process as soon as a sufficient number of pixels has been received (generally a full row of sub-apertures), this mode of testing is referred to as pipeline.

Here we define the Wavefront Processing time (t_{wp}) as the time taken by the image pre-processing (i.e. calibration) and CoG calculation. The Wavefront processing Delay (t_{wd}) is the time between the last pixel read-out to the last slope calculated (i.e. wavefront processing pure delay).

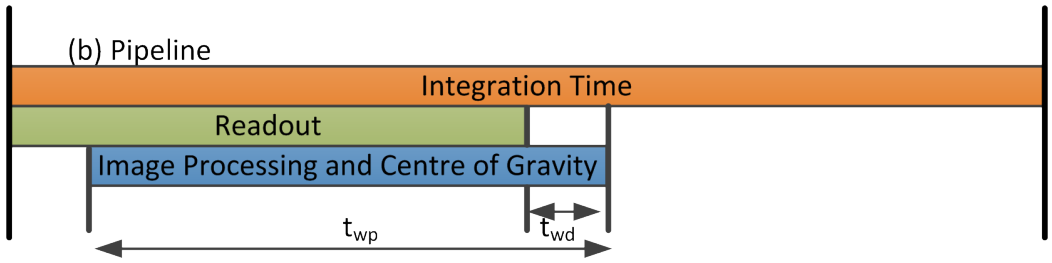


Figure 6.16: A simplified timing diagram showing the pipeline testing. t_{wp} Wavefront Processing time: time taken by the image pre-processing (i.e. calibration) and CoG calculation. t_{wd} Wavefront processing Delay: time between last pixel read-out and last slope calculated (i.e. wavefront processing pure delay).

In an actual AO RTC system the wavefront camera will be streaming pixels into the WPU, in our case the TILE-Gx. Unlike the full-frame testing where the communication delay between camera and WPU was neglected, here we stream data from the FPGA pixel emulator. Our test set-up is shown in figure 6.8.

Wavefront images generally are too big to be sent over Ethernet in a single packet. Each frame will need to be split into many Ethernet packets. Each image frame is sent over Ethernet as quickly as the FPGA Pixel Emulator allows, but a frame delay (t_{fd}) is inserted between frames to ensure the TILE-Gx has finished processing each frame before receiving the next. This method can be seen in figure 6.17. All other timings are defined in section 5.1.2.

Here we define the Read-out time (t_{ro}) as CCD read-out time (i.e. time it takes to read-out all detector pixels). Frame Delay (t_{fd}) as the time between last pixel of one frame and first pixel of the next being sent and Wavefront processing Delay (t_{wd}) as time between last pixel read-out at last slope calculated (i.e. wavefront processing pure delay).

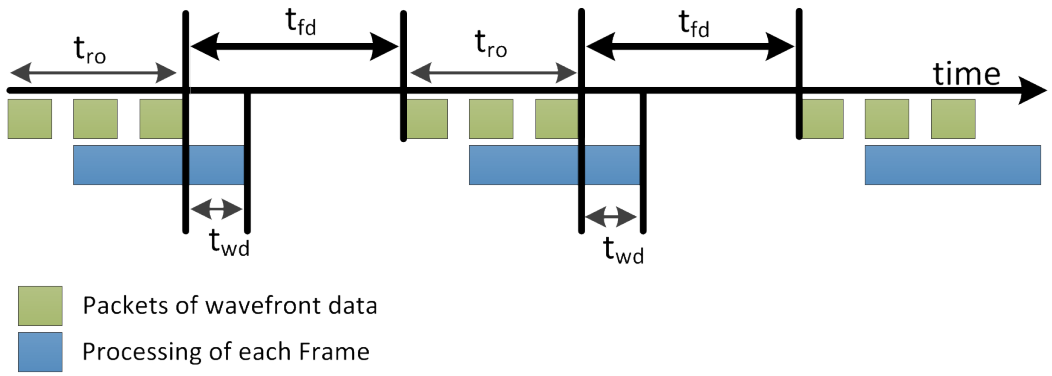


Figure 6.17: A simplified timing diagram showing the data being received on the TILE-Gx. t_{ro} is the Read-out time: CCD read-out time (i.e. time it takes to read-out all detector pixels). t_{fd} is the Frame Delay: time between last pixel of one frame and first pixel of the next being sent. t_{wd} Wavefront processing Delay: time between last pixel read-out at last slope calculated (i.e. wavefront processing pure delay).

6.4.1.2 Sequence diagram

The sequence diagram of the code for the pipeline tests is presented in figure 6.12. The code is very similar to the one used for the full-frame testing (section 6.3), with the distinction that data is being transmitted to the TILE-Gx from the pixel emulator. As with the full-frame testing the main thread is the only thread not running in ZOL mode. The TILE-Gx requires as least one tile to be in non-ZOL mode in order to run the OS. The control thread does not do any processing but supervises the worker threads of which there are 34 (reserving a core for OS activities). Only a single worker thread is represented in figure 6.12.

Each image frame is split across multiple Ethernet packets. As each packet is received, it is checked to make sure that it is the expected packet and that no packets have been missed. A timing is taken when the first packet and last packet of each image frame arrives. Another timing is taken when the whole frame has been processed. With these

three timings we can measure the pixel read-out time (t_{ro}) as well as the pure wavefront processing delay (t_{wd}).

When a packet arrives on the control thread a broadcast message is sent to all worker threads saying there is work to be done. Each worker thread has an assigned list of sub-apertures that it is possible to process when a data packet arrived, once it has processed them it will signal the control thread. Once all worker threads have signaled the control thread the processing has been completed.

Again, as with the full-frame testing the timings are saved in memory while the program is running. It is only once the entire testing procedure is finished that timings are written to disk. This is done to ensure that measurements are not affected by the overheads sustained by writing to disk.

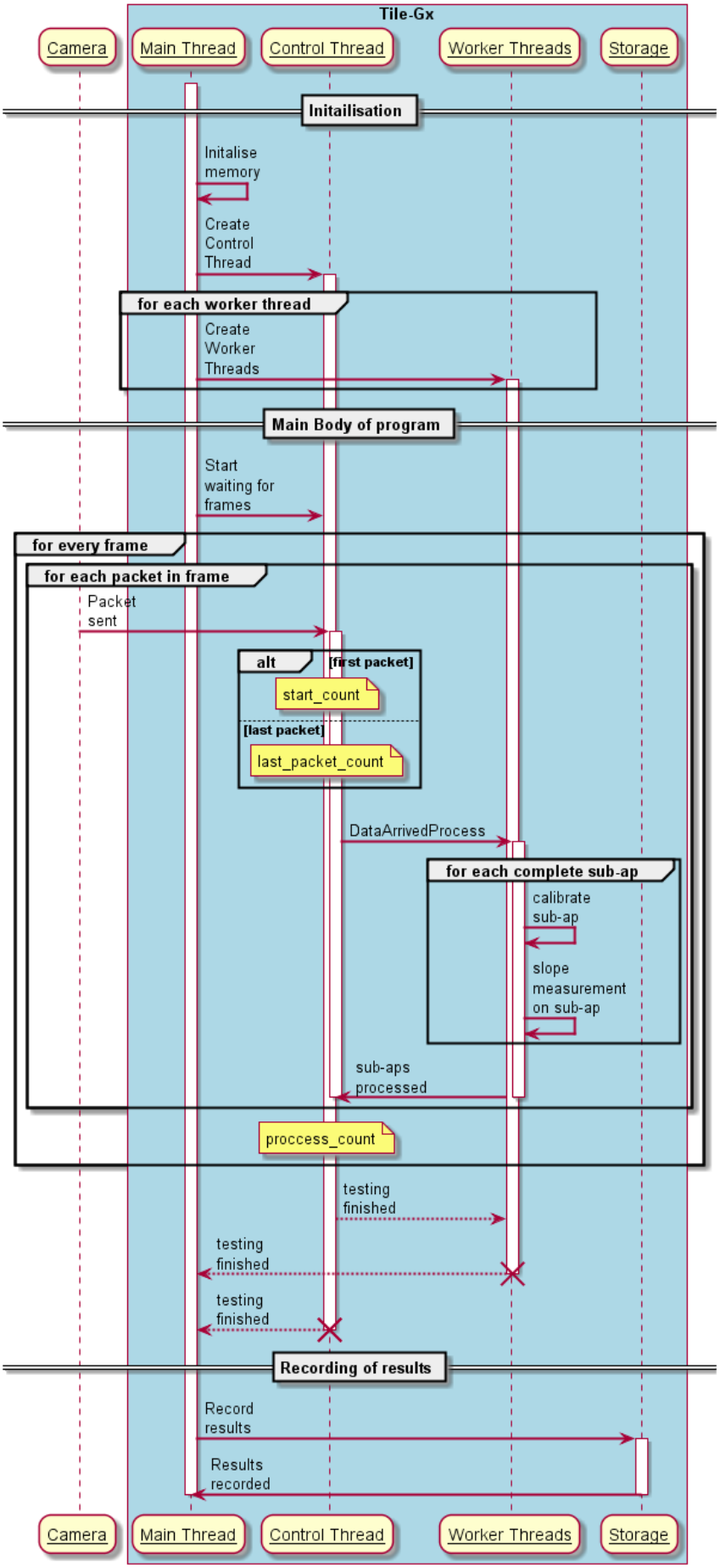


Figure 6.18: A simplified sequence diagram of the pipeline testing.

6.4.1.3 CoG algorithm

In this section, we are only testing the CoG algorithm. The processing remains the same as the one performed with the full-frame processing. In section 6.1.5.1, we have demonstrated that the difference in execution time between the different algorithms tested is minimal. The results obtain here can therefore be extended to other algorithms (such as WCoG and matched filter). A shared library has been created and shared between programs to reduce development and maintenance time.

6.4.2 Mean wavefront processing time

Figure 6.19 shows the mean wavefront processing time t_{wp} (time taken by the image pre-processing (i.e. calibration) and centre of gravity calculation) as a function of the linear detector size N. We can see a similar result to the full-frame testing (see figure 6.13).

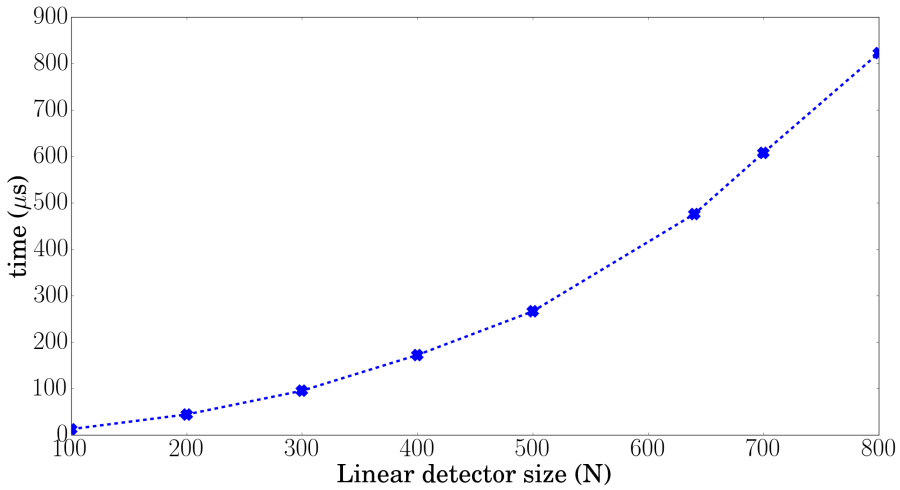


Figure 6.19: Mean wavefront processing time t_{wp} as a function of linear detector size N (the full detector size being N×N). The blue dots represent different combinations of number of sub-apertures and number of pixels per sub-aperture.

6.4.3 Sampling frequency

To put the timings into perspective, results have been plotted as sampling frequency in figure 6.20. The sampling frequency is the inverse of the mean wavefront processing time ($\frac{1}{t_{wp}}$). This allows us to estimate the maximum theoretical sampling frequency the TILE-Gx can function for a given size of detector. For example, a first light E-ELT instrument such as MAORY[5] may require a WFS with 800×800 pixels running at a sampling frequency of up to 1 kHz. Figure 6.20 and table 6.12, shows that the TILE-Gx is able to perform above the required frequency. In addition, and as shown in section 6.1.5.1, adding more cores to the TILE-Gx can very easily be used as a mean to increase performance beyond those observed here.

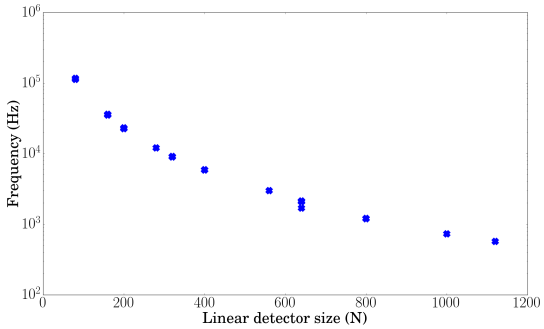


Figure 6.20: Mean sampling frequency ($\frac{1}{t_{wd}}$) as a function of linear detector size N (the full detector size being N×N). The blue dots represent different combinations of number of sub-apertures and number of pixels per sub-aperture.

Table 6.12: Mean sampling frequency (values extracted from figure 6.20).

Detector	frequency (Hz)
280 × 280	12020
400 × 400	5850
640 × 640	2068
800 × 800	1200
1000 × 1000	728

6.4.4 Stability of the execution time

As we showed in the full-frame testing (section 6.3), taking advantage of the ZOL mode of operation on the TILE-GX allows us to achieve near real-time performance. Figure 6.21a, 6.21b and 6.21c shows a

typical histogram of the wavefront processing time (t_{wp}), for systems with detector sizes of 100 × 100, 640 × 640 and 800 × 800 respectively. We see that with the increase in system size comes with an increase in variation in the wavefront processing time (t_{wp}). This can be seen in both the increase in standard deviation and range, but also the loss of the near Gaussian shape that is present in the 100 × 100 case.

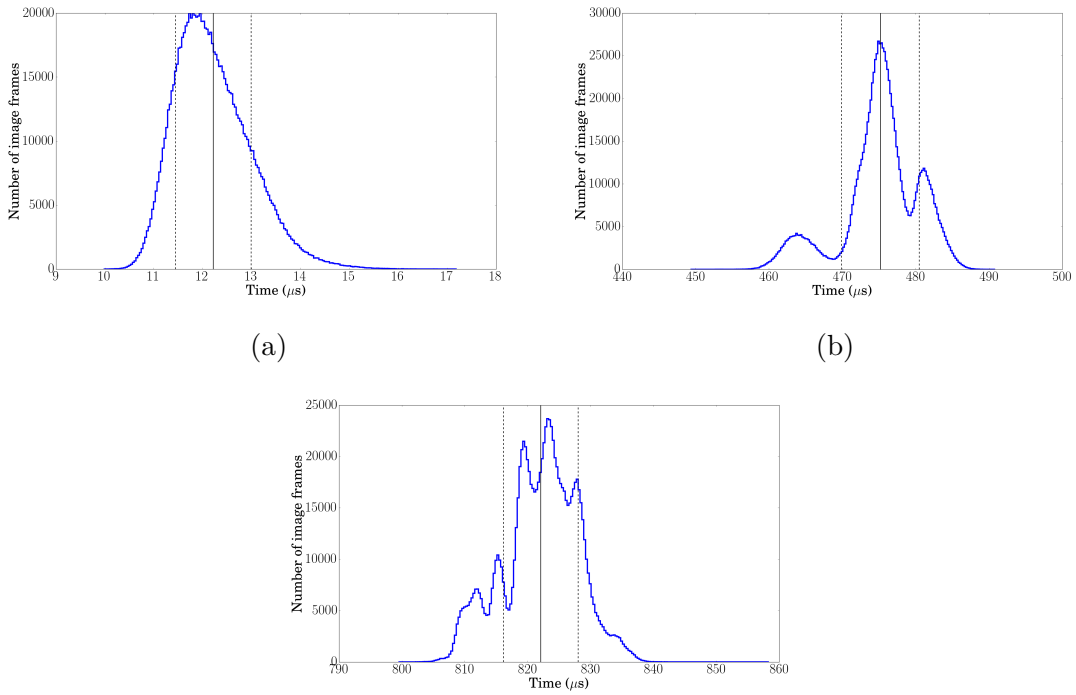
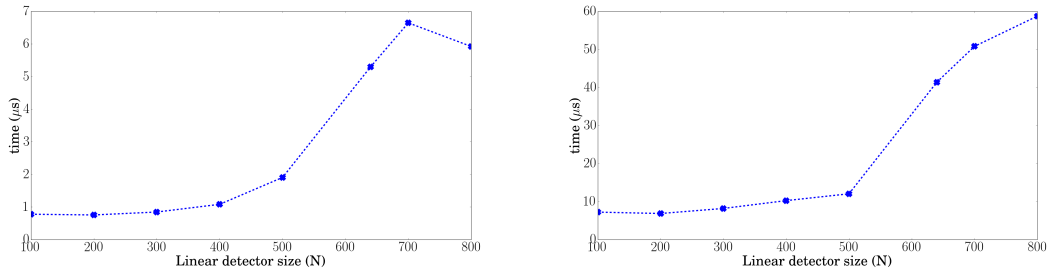


Figure 6.21: Variation in execution time for a selection of detector sizes using the CoG algorithm. Distribution calculated for 10^6 frames. The black vertical line represents the mean and the dashed vertical lines the standard deviation. Figure 6.21a 100 × 100, figure 6.21b 640 × 640, figure 6.21c 800 × 800

In figure 6.22a and figure 6.22b the variation in the overall wavefront processing time (t_{wp}) is presented. We present both the standard deviation as well as the range[4] for a series of detector sizes. An increase in both standard deviation is seen as the overall detector size is increased. This variation is larger than the variation seen in the full-frame testing

[4]Range = max - min

(see figure 6.14) though still small.



(a) (b)

Figure 6.22: (a) Standard deviation and (b) range (Max - Min) of the wavefront processing time (t_{wp}) as a function of linear detector size N (the full detector size being N×N).

6.4.5 Pure wavefront processing delay

The wavefront processing delay (t_{wd}) is the time from the last pixel arriving at the TILE-Gx to the final slope calculated for a given frame (figure 6.16). To calculate this we use equation (6.1) where t_{wp} is the wavefront processing time and t_{ro} is the read-out time for the wavefront camera.

$$t_{wd} = t_{wp} - t_{ro} \tag{6.1}$$

The wavefront processing delay (t_{wd}) is important as it should not depend on the read-out time (t_{ro})[5]. In these tests the data is being sent as fast as possible. A real wavefront camera is likely to have a lower data rate as it is tied to the capabilities of the detector electronics.

Figure 6.23 shows how the mean wavefront processing delay (t_{wd}) increases with the size of detector. For small sizes with a small number of pixels per sub-aperture, each packet may contain a entire row of sub-apertures so the processing is distributed through the whole read-

[5]This requires that t_{ro} is larger than the processing time for the same detector in full-frame mode.

out time (t_{ro}). As we move into the large detector domain each set of sub-apertures will be split across multiple packets so there is a delay in the processing until the last packet arrives allowing the last row of sub-apertures to be processed. This can be observed in figure 6.23 by the steep increase in mean wavefront processing delay for detector sizes 500×500 and larger.

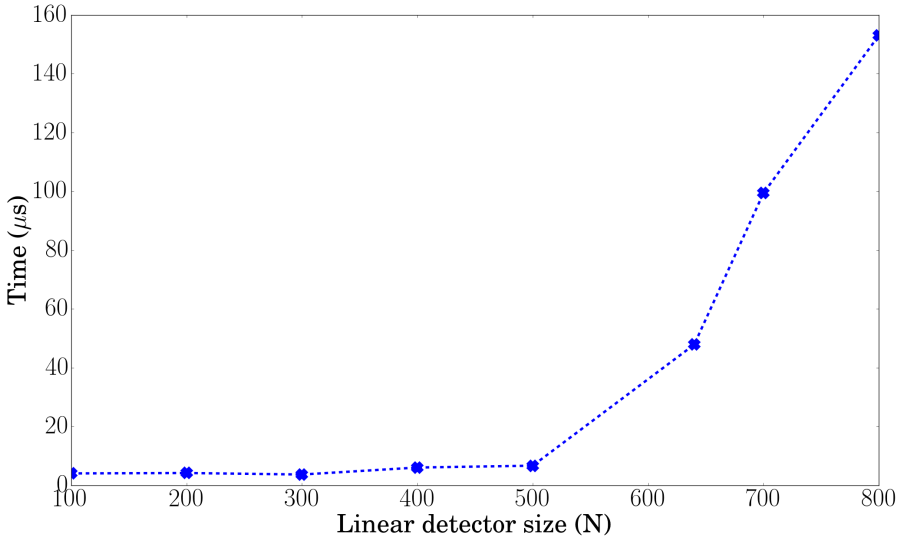


Figure 6.23: The mean wavefront processing delay (t_{wd}) as a function of linear detector size N (the full detector size being N×N).

Unlike the full-frame testing where the detector size was the only factor contributing to performance, in pipeline testing detector size but also the the distribution of sub-apertures and number of pixels per sub-aperture are likely to affect performance. This is due to how the sub-apertures are distributed throughout the data packets being sent. If the last packet of a frame completes a large number of sub-apertures, each with a large number of pixels per sub-apertures, the wavefront processing delay (t_{wd}) is going to be long. If the final packet of data completes a small number of sub-aperture or sub-apertures with low number of pixels, the wavefront processing delay (t_{wd}) is going to be

small. For each of these cases if the detector size is the same, the read-out time (t_{ro}) is going to be the identical. The only difference is the wavefront processing delay (t_{wd}).

When the system size is small ($N < 500$), the TILE-Gx processing time is similar to that of the camera read-out (t_{ro}), this causes only small delays. As the system size increase past this threshold ($N > 500$) the processing takes longer than the camera read-out and we see the wavefront processing delay increasing. This is likely to affect not just the mean but the variance as well. Figure 6.22a and 6.22b we see a sharp increase in both standard deviation (jitter) and range. With the TILE-Gx pushed to its limits, no longer able to keep up with the camera, the variance in processing time increases.

6.5 Competitors and similar products

Mellanox are not the only company that is producing processors that fill the niche of many-core processor with high I/O capabilities. For example, KalRay, a small French company, have started to manufacturing similar chips. Table 6.13 compares a selection of Mellanox's competition with the TILE-Gx. All these processors are recent addition to the market and fill the same or similar roles to the TILE-Gx.

High performance computers typically have large power requirements. The highest single cost of running data centres around the world is electricity[9]. To combat this, lower power chips, such as ARM processors, are being developed to bring these costs down. Some of the processors shown use ARM processors or similar technologies to try and reduce the power consumption of the cards and reduce running costs for their customers.

All these processors support 10 GbE interfaces, while some such as the MPPA-256 and ThunderX support 40 GbE. The type of interface required will be determined when the final WFS designs are made available. It is likely that it will be either 10 or 40 GbE. Alternatively, one could use converters (such as a Camera Link to GigE converter) to support different interfaces with the camera.

The X-gene 3 from Applied Micro has 64 cores all at the high clock frequency of 3 GHz. This card was compared to the Xeon E5 and Xeon D 1540 by the Linley group and they showed that it had a higher memory bandwidth than the Xeon E5[10], rivalling that of the Xeon Phi 5110p[11]. With 64 cores high memory bandwidth and high clock frequency this card may be useful as the front-end of an AO RTC system, as the WPU. The X-gene may also be a promising candidate to perform the Matrix-Vector Multiplication (MVM) in the wavefront reconstruction unit.

Many of these cards were not available when the TILE-Gx was released and are illustrative of how quickly the computer industry is moving.

Table 6.13: A comparison of competitors to the TILE-Gx processors[1, 12, 13, 14, 15]

Manufacturer processor	Release	Cores	GHz	I/O	Power(Watts)
Mellanox TILE-Gx36	2013	36	1.2	4 x 10 GbE	28
Kalray MPPA2®-256-N	2014	256	0.6	2 x 40 GbE or 8 x 10GbE	
Cavium ThunderX	2016	48	2.5	10 and 40 GbE	
Applied Micro X-Gene 3		64	3	4 10 GbE	160
Intel Xeon-D 1540	Q1 2015	8	2.6	2 x 10 GbE	45 W

6.6 Conclusions and perspectives

In an effort to down-select suitable computational technologies for the E-ELT AO modules, in this chapter we have investigated a novel hardware that has the potential of being used as a WPU in AO. We presented real-time performance results for the Mellanox TILE-Gx, a technology offering a high number of cores and multiple 10 Gbps Ethernet ports. Using the TILE-Gx ZOL operating mode, the card reduces interrupts, minimizing both calculation time and jitter and supporting near real-time performance.

Using COTS hardware has a major advantage over ad-hoc hardware, such as the FPGA system designed for SPARTA[16]. Although it requires linking AO hardware solutions to a specify manufacturer and in particular to its development road map, we believe that COTS offers a quicker development time at a lesser cost, while keeping good

maintainability and scalability.

We tested two different modes of operations. The first (i.e. full-frame) waits for all pixels of an entire frame to arrive before processing the data. This is particularly relevant for pyramid WFSing or when using some type of detectors (e.g. IR devices subject to Fowler sampling). The second (i.e. pipeline) starts processing data as soon as a sufficient number of pixels has been received. This mode is particularly relevant for SH-WFSs. In addition, we investigated different CoG algorithms and demonstrated that performance is very comparable in terms of calculation time for all of them.

The results show a very encouraging mean calculation time and very little overall jitter, both for the full-frame and pipeline cases. As an example, performing the wavefront processing calculations using a 240 $\times$240 detector, the TILE-Gx36 only adds 86 μs (resp. 7 μs) after the last pixel has been read in full-frame mode (resp. pipeline mode). When using a 800 $\times$ 800 detector these values increase to 764 μs (and resp. 150 μs). In this paper we have defined jitter as the standard deviation. For these two detector sizes, jitter is respectively 1.27 μs (resp. 2.76 μs for pipeline mode) for a 240 $\times$ 240 detector and 0.84 μs (resp. 5.92 μs for pipeline mode) for a 800 $\times$ 800 detector. In addition, we have demonstrated the very good scalability of the mean execution time: multiplying the number of cores by 2 basically reduces the calculation time by a factor 2.

For small systems ($N < 500$) the TILE-Gx36 is able to process the wavefront data at similar rates to the read-out of detector data. This leads to low ($< 20\mu s$) pure wavefront delays. In these cases the TILE-Gx36 can be used in the front end of AO RTC and add only a small

latency while performing the require computation. As the read-out time becomes larger ($N > 500$) the TILE-GX is no longer able to process the data in the same time and the pure wavefront delay become much larger and the jitter is increased. The use of a TILE-Gx with more cores (i.e. 64 or 72) would allow the WPU to keep up with the read-out for larger detector sizes.

The current baseline for the E-ELT is to use a combination of SH-WFS and PYR-WFS. While the the PYR-WFS are less able to take advantage of the pipeline method described (see section 6.4) (processing cannot begin until over half of the pixels have arrived) the advantage of the PYR-WFS allows less pixels to be used. As we have shown the processing time is proportional to the detector size, a PYR-WFS requiring less pixels would also reduce the computational load.

The overall performance of the current generation of TILE-Gx processors makes it a strong contender for WFS pixel processing unit for most first-light ELT instruments. The next generation are likely to offer increased performance making it a potential contender for all E-ELT scale AO instruments.

References

[1] Mellanox. The tile-gx36 processor product brief. Brochure, 2016.

[2] John D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25, December 1995.

[3] Intel. Intel core i5-4690k processor. Product Brief, 2016, (Accessed: 2016-06-17).

[4] TILERA. Multicore development environment optimization guide. User Guide, 2011.

[5] Emiliano Diolaiti, A Baruffolo, M Bellazzini, V Biliotti, G Bregoli, C Butler, P Ciliegi, JM Conan, G Cosentino, S D'Odorico, et al. Maory: a multi-conjugate adaptive optics relay for the e-elt. *The Messenger*, 140:28–29, 2010.

[6] EZChip. Tile-mx multicore processor. Brochure, 2015.

[7] John Coughlan, Sam Cook, Chris Day, Rob Halsall, and Saeed Taghavi. The data acquisition card for the large pixel detector at the european-xfel. *Journal of Instrumentation*, 6(12):C12057, 2011.

[8] Ripduman Sohan, Andrew C Rice, Andrew W Moore, and Kieran Mansley. Characterizing 10 gbps network interface energy consumption. In *LCN*, pages 268–271, 2010.

[9] APC. Determining total cost of ownership for data center and network room infrastructure. Technical report, APC, 2003.

[10] Linley Gwennap. "x-gene 3 challenges xeon e5". Technical report, The Linley Group, "2016, (Accessed: 2016-06-17)".

[11] Intel. Intel xeon phi coprocessor 5110p, November 2012.

[12] kalray. Mppa2Âő-256-n bostan networking. Brochure, 2016, (Accessed: 2016-06-17).

[13] Cavium. Thunderx family of workload optimized processors. Product Brief, 2016, (Accessed: 2016-06-17).

[14] Applied Micro. X-gene world's first armv8 64-bit server on a chip solution. Product Brief, 2016, (Accessed: 2016-06-17).

[15] Intel. Intel xeon processor d-1540. Product Brief, 2016, (Accessed: 2016-06-17).

[16] SJ Goodsell, E Fedrigo, NA Dipper, R Donaldson, D Geng, RM Myers, CD Saunter, and C Soenke. Fpga developments for the sparta project. In *Optics & Photonics 2005*, pages 59030G–59030G. International Society for Optics and Photonics, 2005.

Chapter 7

Wavefront reconstruction: a memory bandwidth limited problem

Parts of this work have been published in SPIE astronomical telescopes and instrumentation 2015, Monthly Notices of the Royal Astronomical Society and presented at Adaptive optics for ELTs 4.

- *David Barr, Alastair Basden, Nigel Dipper, Noah Schwartz, Andy Vick, and Hermine Schnetler. "Evaluation of the Xeon Phi processor as a technology for the acceleration of real-time control in high-order adaptive optics systems." In SPIE Astronomical Telescopes+ Instrumentation, pp. 91484B-91484B. International Society for Optics and Photonics, 2014.*

- *David Barr, Alastair Basden, Nigel Dipper, and Noah Schwartz. "Reducing adaptive optics latency using Xeon Phi many-core processors." Monthly Notices of the Royal Astronomical Society 453, no. 3 (2015): 3222-3233.*

- *David Barr, Alastair Basden, Nigel Dipper, and Noah Schwartz.*

Reducing adaptive optics latency using many-core processors. Proc. AO4ELT4, 1, 2015.

Contents

Wavefront reconstruction, translating measured wavefront slopes into new deformable mirror (DM) commands, is by far the most computationally intensive algorithm that an ELT-scale real-time control (RTC) system is required to perform. The most common wavefront reconstruction algorithm used is the Matrix-Vector Multiplication (MVM). The DM commands d are related to the slopes s through a linear equation $d = G^{-1}s$, where G^{-1} is the control matrix. The computational complexity for an MVM grows as $\mathcal{O}(M^2)$, where M is the number of degrees of freedom of the AO system. A simple extrapolation of current hardware to the ELTs is not sufficient and new solutions need to be investigated.

In this chapter, we investigate the performance of the Intel Xeon Phi for wavefront reconstruction using the MVM algorithm. We first start with a brief reminder of wavefront reconstruction related hardware issues. We then describe specific details of the Xeon Phi itself. We focus on its architecture and intrinsic performance. Finally, we detail benchmarking results, concentrating on accurate time measurements in a wide range of parameter space.

7.1 Introduction

In recent years, several novel computationally efficient wavefront reconstruction algorithms have been developed[1, 2, 3]. These alternative wavefront reconstruction approaches are typically iterative and generally unable to efficiently take advantage of modern multi-core and many-core hardware architectures. Although the MVM typically has the largest requirements in terms of number of operations and memory usage compared to these other methods, it is highly parallelisable mak-

ing efficient use of modern multi-core architectures and widely used by the AO community. For small AO systems, CPU based systems can typically be used. As we move towards ELT-scale systems, calculations become more and more difficult to handle with CPU alone, limited both by available memory bandwidth, and raw processing power.

To achieve the required computational power, many groups have focused on GPUs ([4, 5, 6, 7, 8, 9]) since they offer a potential suitable parallel environment to reduce the latency associated with the MVM calculation. FPGAs have also been used, although typically for smaller systems[10], for only a section on the AO control loop[11], or limited to the pixel processing as part of heterogeneous RTC hardware[12]. These hardware accelerators generally suffer from the same disadvantages: limited data transfer into and out of the accelerator. They lead to complex heterogeneous computing environments which give rise to complex memory structures and the movement of large quantities of data between different computational components. Accelerator architectures traditionally evolve quickly as new hardware is released, which may not be compatible with older systems, leading to lifetime and portability issues. This can cause long development times and difficulty in maintaining and upgrading systems.

The algorithm we have decided to investigate with the Xeon Phi is the MVM due to the widespread use and ease of parallelisation. An overview of the MVM as well as other algorithms can be found in section 5.3. The complexity of the MVM is not in the difficulty of implementation but in the high number of calculations needing be performed, which is why hardware accelerators need to be investigated (see 5.3.1.1).

The Xeon Phi uses x86 instruction set microprocessors (same as

conventional CPUs), which may help in lowering the barriers to entry compared with GPUs or FPGAs, i.e. no specialised code base or API is required. The implemented code can easily be modified and upgraded should a more performant hardware be released. The Xeon Phi also offers high memory bandwidth to accelerate memory-bound parallel algorithms. It is however, designed for high-performance computing where the requirements are more focused on the mean execution time rather than on the determinism of execution time. A detailed analysis of performance in a realistic AO environment is therefore essential.

Previous investigations were limited to non-real-time (non-RT) Linux systems[13] or focused on a very specific AO system[14] making the generalisation to other systems difficult. Our approach here, is not to concentrate on a particular AO design, but rather to investigate a wide range of parameter space. In addition, a detailed analysis of the timings is crucial to fully understand the limitations of the hardware and extrapolate to future hardware developments. Different science cases will have different tolerances on the acceptable jitter (variation in execution time) or outliers (results significantly apart from the mean) for example, which may or may not impact science results significantly.

7.2 Intel Xeon Phi

The Xeon Phi is a many-core accelerator co-processor card connected to a processor via a PCIe bus offering a high level of programmability (standard C/C++ with compiler assisted offload), high throughput, high performance per watt and low cost. The main disadvantage, as with most accelerators, is that data communication between the host computer and the Xeon Phi will add unwanted delays (and jitter) to

the AO loop. This leads to a heterogeneous computing environment which may cause issues with complex memory management and makes optimisation difficult. The Xeon Phi is similar in that sense to general purpose GPUs used in high-performance computing environments. The Xeon Phi differs however from GPUs by offering x86 instruction set cores, allowing programmers to use the same design techniques as they would with CPUs. This has the potential to speed up development time and does not require specialist knowledge of programming paradigms and toolkits such as CUDA or OpenCL.

7.2.1 Xeon Phi architecture

The Xeon Phi model under investigation is the 5110P, which offers 60 cores, a clock frequency of 1 GHz, 8 GB of GDDR5 memory and has a maximum theoretical memory bandwidth of 320 GB s^{-1}. The specifications of the Xeon Phi 5110P are summarised in table 7.5. The Xeon Phi 5110P clock frequency is slower than that of modern CPUs which can typically reach 3-4 GHz. This suggests that the Xeon Phi would be unable to compete for performance on sequential code. Given the number of cores and the high memory bandwidth, it has the potential to outperform current CPUs on parallel codes such as the MVM.

Table 7.1: The specifications of the Xeon Phi 5110P[15]

Processor Number	5110P
Number of Cores	60
Clock Speed	1.053 GHz
Cache	30 MB
Max Memory	8 GB
Max Memory Bandwidth	320 GB s^{-1}

7.2.1.1 Memory Hierarchy

The Xeon Phi 5110p has two levels of cache (L1 and L2), and it also has 8 GB of GDDR5 memory. L1 cache has a 32 KB instruction cache and 32 KB data cache. This cache has an access time of 1 clock cycle. This means that once data is in this cache, the data can be used by the processor on the next clock cycle.

The L2 cache is globally shared, with each core contributing 512 KB, this means the processor has ≈30 MB of L2 cache. This cache has an access time of 11 clock cycles and supports hardware prefetching and ECC correction.

7.2.2 Memory Bandwidth

When performing the wavefront reconstruction in an AO RTC using a matrix-vector multiplication algorithm, the input vector is updated at every iteration (typically from hundreds to thousands of times per second), while the control matrix will remain constant for periods of time between tens of seconds to several hours. However, for large AO systems the matrix is too large to be stored in the L2 cache alone. The matrix must be read from the slower GDDR5 memory. Therefore, memory bandwidth from this slower GDDR5 memory becomes a performance limiting factor. CPU-based systems typically have large banks of DDR3 memory which are relatively slow. The Xeon Phi has access to faster GDDR5 and has advertised a maximum theoretical memory bandwidth of 320 GB s^{-1}. In practice the Xeon Phi appears to have a read memory bandwidth of 164 GB $s^{-1}$and a write memory bandwidth of 76 GB s^{-1}[16].

Here, we have measured the MVM computation time, and use this

information to calculate the achieved memory bandwidth. Figure 7.1 shows memory bandwidth as a function of AO size (and the control matrix size stored in memory) for a single Xeon Phi calculated from the total offload time. This offload time includes the data transfer and calculation time, so the memory bandwidth of the Xeon Phi processor alone (i.e. without data transfer) will be slightly higher. For large AO systems, the calculation time is limited by the memory bandwidth, which peaks at about 160 GB s^{-1}, in agreement with[16]. We note that for smaller systems, where the control matrix and both wavefront slope input vector and DM command output vector can fit into cache memory, memory bandwidth is not the performance limiting factor. However, these systems are typically smaller than planned E-ELT AO systems.

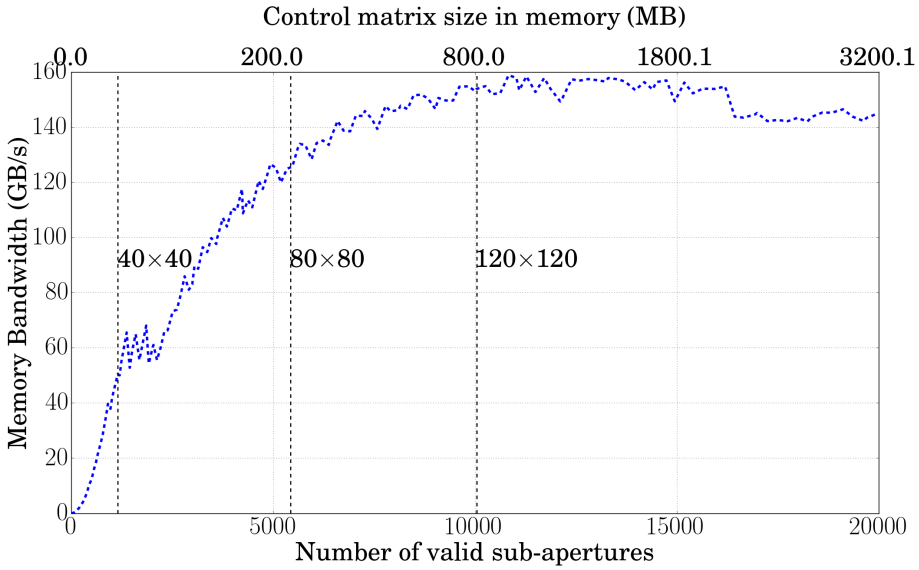


Figure 7.1: Memory bandwidth for a single Xeon Phi performing an MVM as a function of number of valid sub-apertures (bottom) and the size of the control matrix in memory (top). This includes the data transfer time across the PCIe bus. The dashed vertical lines represent approximate AO system sizes (including only the valid sub-apertures due to the shape of the E-ELT primary mirror M1)

When the control matrix is larger than the Xeon Phi L2 cache

(30 MB), we see a drop in memory bandwidth due to the processor having to transfer all or part of the control matrix from the slower GDDR5 memory. As we increase the size of the control matrix the processors have to make more and more calls to the slower GDDR5 memory. This continues until the control matrix reaches around 800 MB where the memory bandwidth levels around 160 GB s^{-1}. At this point, the control matrix is significantly larger than the L2 cache and most memory access is with the GDDR5 memory. The MVM like other BLAS-1[1] or BLAS-2[2] routines is memory bandwidth limited.

To confirm the achievable memory bandwidth, we used the industry standard STREAM memory benchmarking[17] (see section 4.2.3.2) on both the Xeon E5 (the host computer) and the Xeon Phi. We compared the memory bandwidths using a TRIAD[3] test which is the STREAM benchmarking scheme most closely resembling a MVM operation. The Xeon E5 achieved a peak memory bandwidth of 63.7 GB $s^{-1}$and the Xeon Phi of 166.5 GB s^{-1}. Other groups have published similar results[16]. The results of the STREAM tests are presented in table 7.2.

The multiplication of a matrix by a scalar (i.e. scale test) and addition of two matrices (i.e. add test) are consistent with the Triad test. It is only for the copy that the memory bandwidth of the Xeon Phi is slightly lower (approximately 140 GB/s). The copy test involves a single read and a single write, whereas the other tests include some form of calculation. The lower write memory bandwidth has therefore more impact for this type of test.

[1]Vector-Vector operations

[2]Matrix-Vector operations

[3]The Triad test involves the addition of two vectors (b & c) one vector multiplied by a scaling factor (q) $a_i = b_i + q \times c_i$

Table 7.2: STREAM results of the Xeon E5 and the Xeon Phi.

Tests	Bandwidth (MB s^{-1})	
	Xeon E5	Xeon Phi
Scale	62530.5	161386.1
Copy	62720.4	140675.6
Add	63591.5	166016.9
Triad	63739.9	166016.9

Table 7.3 compares the advertised (theoretical), achieved memory bandwidths using STREAM as well as the percentage of the advertised that was attained. The results are shown for the Dual Xeon E5-2650, the Xeon Phi 5110p as well as the NVidia K40 GPU[18] (a GPU released at around the same time as the Xeon Phi, which enables a direct comparison between hardware of the same generation). It can be seen that although the Xeon Phi has a higher theoretical maximum, the GPU can achieve a higher percentage of the advertised bandwidth than either the Xeon E5 or the Xeon Phi.

Table 7.3: A comparison of advertised and achieved memory bandwidths for Dual Xeon E5-2650, NVidia K40 GPU and Xeon Phi 5110p.

	Dual Xeon E5-2650	NVidia K40	Xeon Phi
Advertised Max. GBs^{-1}	2x51.20	288	320
STREAM GBs^{-1}	63.7	229	166.5
Percentage	62.2 %	79.5 %	52.03 %

7.2.3 FLOPS

As detailed in section 4.2.3.1, the FLOPS are the number of Floating point operations per second a processor can achieve. The Xeon Phi is advertised to be able to reach 1.011 TFLOPS. The equation does not take into account the memory bandwidth of a system. This value is far

larger than we could expect to reach with the MVM or any other BLAS-1 or BLAS-2 operations due to the memory bandwidth limitations of these operations.

Figure 7.2 shows the number of FLOPS that the Xeon E5-2650, Xeon Phi and 2 Xeon Phis used in parallel have achieved when performing the MVM algorithm as function of AO system size. A peak in performance is seen for the Xeon E5-2650 below 2000 sub-apertures, which is when the matrix no longer fits in cache memory of the CPU. Curves for the single and dual Xeon Phi follow similar curves as seen with the memory bandwidth, this is to be expected with a memory bandwidth limited problem.

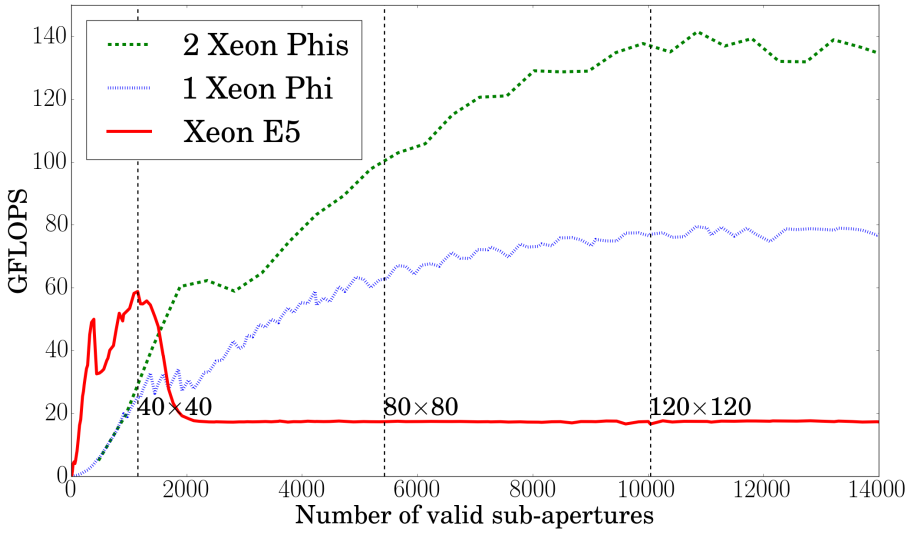


Figure 7.2: Number of FLOPS achieved during MVM for a dual Xeon E5-2650, one Xeon Phi and two Xeon Phis.

Table 7.4 shows the number of FLOPS that the Xeon E5-2650, Xeon Phi and 2 Xeon Phis have achieved when performing the MVM algorithm for a selection of AO system sizes. The AO system sizes take into account the primary mirror mask, removing approximately 25 % of the total sub-apertures (e.g. an 80×80 systems has a total of 4700 valid

sub-apertures) (see appendix A.2). It can be seen that the performance of the Xeon Phi is much lower than the advertised FLOPS. We see that using a second Xeon Phi almost doubles the achievable FLOPS for a large system, in line with expectations.

Table 7.4: Comparison of FLOPS for dual Xeon E5-2650, single Xeon Phi and a dual Xeon Phi system. Max is the maximum GFLOPS achieved across entire tested range, 0-14000 total valid sub-apertures.

GFLOPS	40x40	80x80	120x120	Max
Xeon E5-2650 V2	58.7	17.3	17.3	58.7
1 Xeon Phi	25.2	62.7	76.9	79.4
2 Xeon Phis	29.0	100.3	136.9	141.5

7.2.4 Data transfer

A large disadvantage of using hardware accelerators such as the Xeon Phis or GPUs is the data transfer between the host and the coprocessor. Data arrives on the host computer, then is transferred to the coprocessor (e.g. Xeon Phi or GPU), processed on the coprocessor and finally the results are transferred back to the host. This adds an additional stage to the data path which increases the latency of the system.

The Xeon Phi is connected to the host computer via a 16 lane PCIe bus version 2.0. Each lane of the PCIe 2.0 has a data transfer rate of 500 MB s^{-1}; the 16 lanes will have a maximum data transfer rate of 8 GB s^{-1}. As with the memory bandwidth, the maximum rate stated may not be achievable in reality. The overall size of the data per frame to be transferred (slopes and DM commands) adds up to approximately 38 KB for this case.

Figure 7.3 shows the data transfer rate that can be achieved to and from the Xeon Phi. Measurements are obtained by simply timing the

transfer of a square matrix to and out from the Xeon Phi. The Xeon Phi data rates are low for small sizes (< 10 MB); this is likely due to the high amount of overhead that are included with the transfer of data. As the matrices increase in size, the data rate quickly plateaus at close to 1.7 GBs^{-1}. This is much lower than the possible 8 GB s^{-1}from a 16 lane 2.0 PCIe bus.

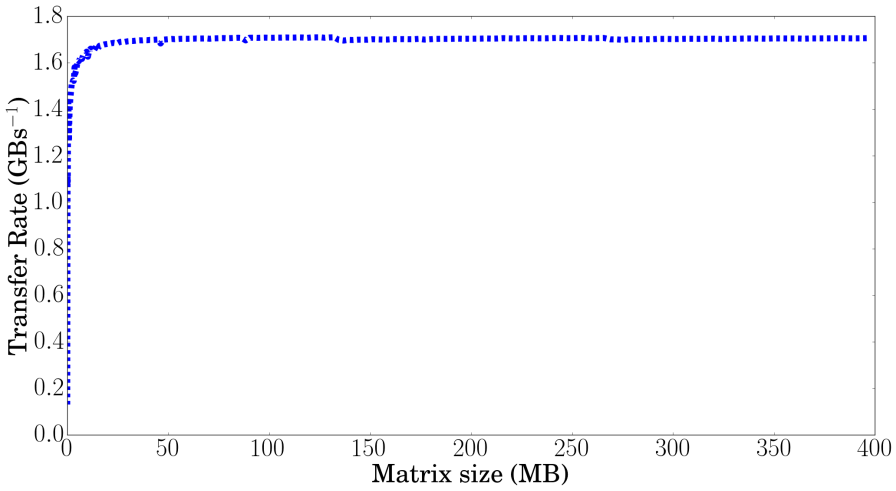


Figure 7.3: Data transfer rates between host and the Xeon Phi as a function of the size of the data being transfered.

For the MVM we are only uploading a wavefront slope vector to the Xeon Phi and downloading a DM command vector. For a SCAO system with a single deformable mirror, we can assume the wavefront slope vector (containing both X and Y slopes) to be twice the size of the DM command vector. For example, an 80×80 system will have 12800 slopes in X and Y, and for an E-ELT aperture and only accounting for the valid sub-aperture, may have approximately 9440 valid slopes.

This is much lower than the values shown in figure 7.3 which are more applicable to the upload of the control matrix.

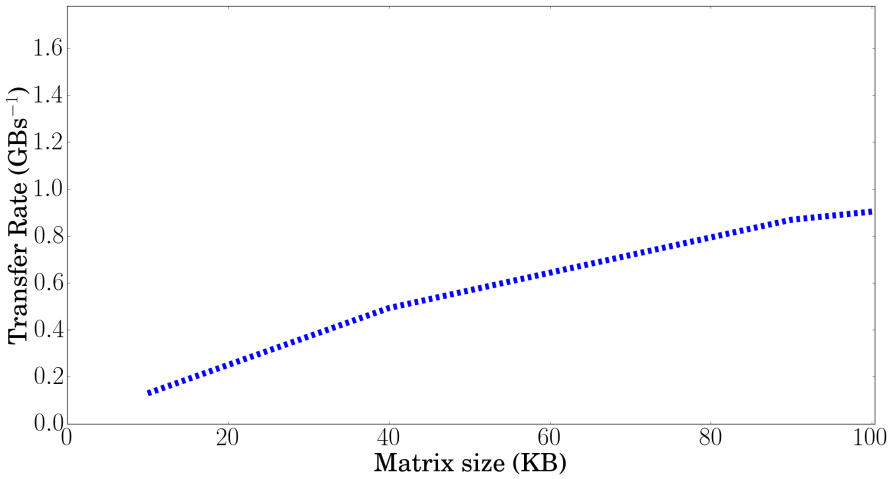


Figure 7.4: Data transfer rates between host and the Xeon Phi as a function of the size of the data being transferred. Zoomed to show size of vectors being used in AO RTC.

Data transfer rates for data sizes closer to what would be typically seen for transferring slope and DM command vectors into the AO RTC is presented figure 7.4. It can be seen that the values measured are much lower than the 1.6 GB s^{-1}shown in figure 7.3. For sizes of 38 KB suggested for an 80 $\times$ 80 sub-aperture system the data transfer is 0.4 GB s^{-1}. This much lower data transfer rate is likely to be caused by the overheads. These overheads are likely to be constant or slightly increasing as the data size being transferred is increased. This overhead causes the small sizes to be influenced by the overhead much more than the larger sizes.

7.2.5 Developing on the Xeon Phi

There are two modes of using the Xeon Phi: the native mode and the offload mode. The native mode refers to the use of the Xeon Phi as a standalone computer. This is the simplest method of using the Xeon Phi where it can be accessed remotely by using SSH. This is not however

a mode that will be used here.

The offload mode uses the Xeon Phi as an extension to the computer similar to how GPUs work. This means that large numerical computations can run on the many cores of the Xeon Phi while the host is performing other tasks.

7.2.5.1 Offload modes

There are three modes of offload and are listed below.

- Automatic offload
- Compiler assisted
  - Implicit
  - Explicit

The automatic offload mode allows for certain function to be automatically offloaded to the Xeon Phi. This will only work on certain functions such as those in the Intel Math Kernel Libraries (MKL). A flag is set for the compiler to decide, based on what is expected of the size of computation, whether these functions should be offloaded. This is the quickest method of development to use the computing power of the Xeon Phi, but not necessarily the most efficient. The program will upload all the required data to and from the Xeon Phi for these functions. This means that a large amount of data is being transferred.

In the compiler assisted modes the decision on what parts of the code are offloaded is left to the developer. In the implicit mode, the developer marks certain memory as 'shared' and at runtime the program will automatically synchronise the data between the host computer and the Xeon Phi. The shared data must be set as global using Intel Cilk Plus

API available only for C/C++. Cilk Plus is an extension to C/C++ to assist in developing task for multi-core or vector processing. The shared data will be synchronised at runtime and supports complex data structures. Functions can also be offloaded to the Xeon Phi by marking them as shared. This method does not allow blocks of code to run on the Xeon Phi, only functions. At run time, the computer decides what data is needed and when functions can be offloaded; this is not directly controlled by the developer. The method for using this is shown in listing 7.1.

Listing 7.1: Implicit memory offload example using Cilk Plus.

```
// Declaring shared variable
int _Cilk_shared sharedInt;
// Declaring shared function
int _Cilk_shared MySharedFunction(


...
// Offload function to the Xeon Phi
mySharedInt = _Cilk_offload mySharedFunction(
```

The last offload mode is called explicit where the developer has explicit control over all data transfers between the Xeon Phi and host, as well as code execution. The keyword `#pragma` is used to indicate regions of code that will be offloaded to the Xeon Phi. An example is shown in listing 7.2. In this example a, b and c are all arrays of length N. Arrays b and c are transferred to the Xeon Phi asynchronously; this does not block the host thread. Array a is transferred to the Xeon Phi and a simple calculation is performed. Array c is transferred back to

the host and all memory is freed on the Xeon Phi. This section does block the host thread, though can be identified as non-blocking and performed asynchronously.

Listing 7.2: Explicit memory offload example.

```
#define ALLOC alloc_if(1) free_if(0)
#define FREE alloc_if(0) free_if(1)
int * a = malloc(N*sizeof(int));
int * b = malloc(N*sizeof(int));
int * c = malloc(N*sizeof(int));

// Transfer some data to Xeon Phi
#pragma offload_transfer(mic:n) /
        in(b) in(c: length(N) ALLOC)



// Transfer some data to Xeon Phi
// and perform some calculation
// output c back to host
#pragma offload target(mic:n) /
        in(a: length(N) alloc_if(1) free_if(1)) /
        nocopy(b: length(N) FREE) /
        out(c: length(N) FREE) in(N)
{
        for(int i =0; i < N; i++)
        {
                c[i] = a[i]+b[i];
        }
```

```
}
```

As can be seen in listing 7.2 the explicit offload is much more controllable then the implicit mode. This works to our advantage, as we can control exactly which and when data needs to be updated on the Xeon Phi. In AO RTC the control matrix is updated at a much lower frequency than the wavefront slope vector. In this case, we can transfer the control matrix to the Xeon Phi and save it. During each loop, we only upload the wavefront slope vector and download the DM commands. For this reason we are going to be using the explicit mode as it give us greater control over the data transfers and code being executed on the Xeon Phi.

7.2.6 Libraries and APIs

There are many libraries and APIs available for developing applications on the Xeon Phi. For example, Open Multi-Processing (OpenMP), (see section 4.3.1.2) a cross-platform API for developing shared memory multiprocessing applications. OpenMP adds a selection of keywords accessed through `pragma`s that enable the developer to quickly develop applications that take advantage of the multiple cores available on modern CPUs.

Another example is Open Computing Language (OpenCL) (see section 4.3.1.3), a cross-platform framework that allows the execution on many different computational hardware such as CPUs, GPUs, Xeon Phis, DSPs and FPGAs. It is intended for hardware independent development that can be then compiled and executed on any hardware available. It is also aimed at parallel processing so is able to target many and multi-core processors.

In a study by Karimi[19] it was found that CUDA outperformed OpenCL on a GPU, both for the data transfer and data processing. It concluded that for high performance or time critical tasks CUDA is the better choice. While OpenCL offers the ability for code to be recompiled for many different hardware architectures large amounts of hardware dependant optimisation as still required. Even with these optimisation OpenCL still has not offered the same performance as native implementations. For all these reasons we have decided for our investigation we are going to focus on the provided development tools.

Open accelerators (OpenACC), (see section 4.3.1.4) is an API designed to simplify the development of parallel programming on heterogeneous hardware. The developer uses keywords to mark regions of code to be offloaded to hardware accelerators. While OpenMP only supports parallelisation across the hardware the code is running on, OpenACC supports the use of hardware accelerators. At the time this work was being carried out on the Xeon Phi a OpenACC compiler was not available, for this reason OpenACC has not been investigated.

7.2.6.1 MKL and MAGMA

Intel's Math Kernel Library (MKL) is a library offering a wide range of optimised maths routines for many applications such as science and engineering. The core of MKL revolves around the standard set of math functions including Basic Linear Algebra SubPrograms (BLAS), Linear Algebra Package (LAPACK), Scalable LAPACK (ScaLAPACK) and Fourier transforms.

Matrix Algebra for GPU and Multicore Architecture (MAGMA) is a project that aims at creating a new generation of linear algebra libraries

with specific design and optimisation for hardware accelerators such as GPUs and Xeon Phis[20].

MKL and MAGMA are standard maths libraries and provide a MVM function optimised for the Xeon Phi. In figure 7.5 we show the comparison of MAGMA and MKL, both performing a matrix-vector multiplication, as a function the matrix size. These results were obtained with a non-RT kernel. Moving to a RT system should not change the mean offload time significantly but should reduce the variations in execution time. MKL outperforms MAGMA for all cases. MAGMA offers much more abstraction from the Xeon Phi architecture, which allows quick development but at the cost of reduced performance.

In-house code using OpenMP was developed and optimised for certain matrix sizes and was able to outperform MKL on some specific cases. In general, MKL gave a high baseline performance for all cases. It was decided to use MKL and focus on general performance of the Xeon Phi rather than focus on specific cases where in-house code can be tuned and optimised to obtain the best performance. Using MKL has the double advantage of reaching very good performance overall but also ensuring that simple software design techniques can be used without requiring in-depth knowledge of the Xeon Phi architecture.

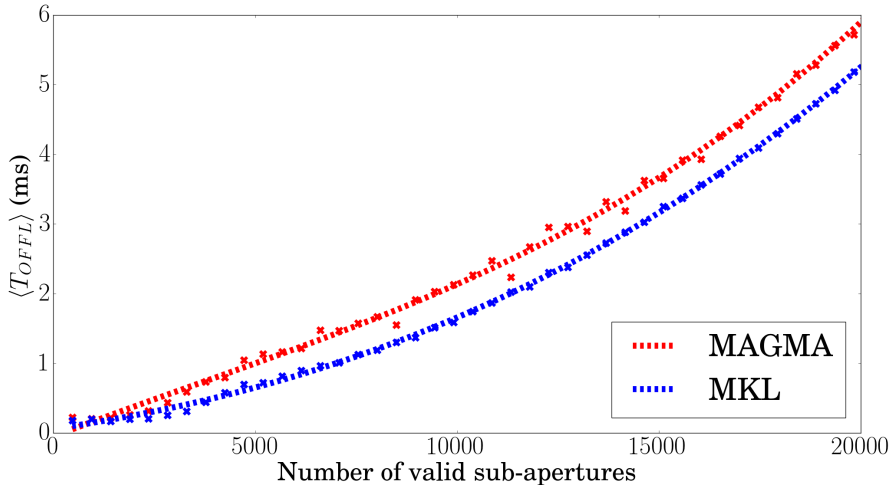


Figure 7.5: Comparison of MAGMA and MKL mean offload time $\langle T_{OFFL} \rangle$ as a function of the number M of valid sub-apertures using 10^4 samples. Red: MAGMA; Blue: MKL. Results obtained using a non RT Linux kernel. Crosses represent actual measured performance and the dashed lines a polynomial fit.

7.2.7 Operating system and real-time

Modern computers are generally not RT processors and operating systems will always have background processes running which affect the determinism of the system. AO systems need a high level of determinism which non-RT operating systems are generally unable to provide. RT Linux on the other hand, gives us greater control on the order (priority) in which processes are carried out by the operating system (see section 4.3.3). These processes with raised priority are able to pre-empt the lower priority tasks to give greater control and predictability in execution time. For the host computer, both a non-RT and a RT Linux kernel will be investigated. A RT pre-empt 3.10 Kernel was installed on the host computer which is running Red Hat 6.4. The RT Linux kernel was not edited nor modified. The Xeon Phi itself is running a non-RT micro-operating system based on a Linux kernel.

7.3 Benchmarking the Xeon Phi

The aim of this section is to investigate, characterise and benchmark the Xeon Phi in the context of a low latency and low jitter AO control loop. The role of the wavefront reconstruction is to take the wavefront slope vector delivered by the wavefront processing unit and calculate the new deformable mirror commands. We do not include here additional tasks (often characterised as soft RT tasks) and do not relate our findings to any specific instrument design, preferring to adopt a more general approach.

7.3.1 Testing architecture

The hardware configuration used in this investigation is shown in figure 7.6. We will be using the Xeon Phi to accelerate MVM calculations and the rest of the AO processing tasks (such as image calibration, slope calculation and DM control laws) are performed by the host computer. We will be investigating a single Xeon Phi as well as a dual Xeon Phi configuration. The host computer, composed of a dual Xeon E5-2650, receives wavefront camera(s) pixels (typically a Shack-Hartmann sensor) and calculates the slopes. The Xeon Phi specifications are shown in table 7.1 and the host specifications are shown in table 7.5.

In the studied configuration, the wavefront camera data is emulated and not physically connected to any hardware, ensuring that the system is not limited by the camera's frame rate. The Xeon Phi then receives the slopes from the host computer through a PCIe bus and computes the command vector that would be sent to the DM(s). Equally, no DM is physically connected to the host PC. The control matrix is stored before calculations in the Xeon Phi's memory, while wavefront slopes

and DM commands vectors are transferred at each AO frame.

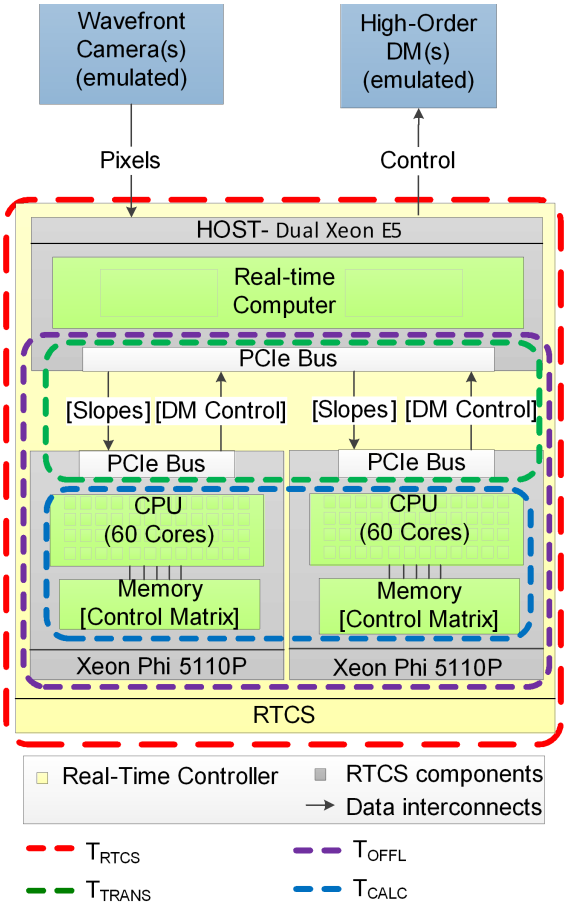


Figure 7.6: Hardware configuration used to benchmark the Xeon Phi. Two Xeon Phis are connected to the host computer (Xeon E5-2650) via PCIe bus. The dashed boxes represent details of the four different times that are investigated.

Table 7.5: Specifications of the Xeon E5 host computer.

	Xeon
Processor	E5-2650
Release Year	2012
#Cores	32
Clock speed	2.0 GHz
L2 Cache	20 MB
Memory type	DDR3
Memory Bandwidth	51.2 GBs^{-1}

7.3.2 Definition of the measured times

The Xeon Phi and the Xeon E5 have a separate clock, making it difficult to accurately time data transfer and MVM calculations using the same clock. In order to produce accurate timings, we measure the difference between times on the CPU and times on the Xeon Phi, removing issues of asynchronous clocks. Throughout this chapter, we will investigate 4 different timings to fully benchmark the performance of the Xeon Phi. Data is taken according the diagram presented figure 7.7.

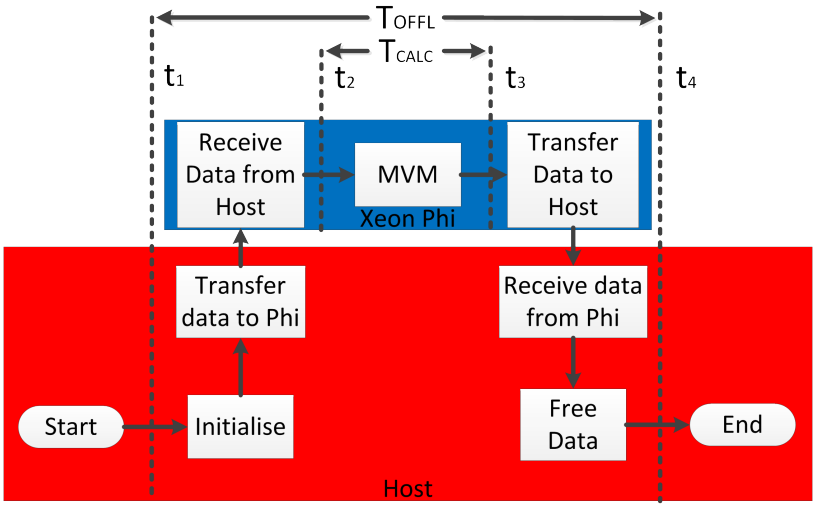


Figure 7.7: Diagram showing a simplified version of the implemented process and the four different timings measured to benchmark the Xeon Phi. t_1 and t_4 are taken on the Xeon E5; t_2 and t_3 are taken on the Xeon Phi. Due to the Xeon Phi and the Xeon E5 having separate unsynchronised clocks, only the total data transfer time is accessible and is calculated from $T_{OFFL} - T_{CALC}$.

- T_{OFFL} is the offload time: this represents the time from the first data sent from the host computer to the Xeon Phi(s) to the last data received back on the host computer. This includes data transfer (i.e. slope and DM command vectors) and MVM calculation on the Xeon Phi(s). $T_{OFFL} = t_4 - t_1$ (see Fig 7.7).

- T_{CALC} is the calculation time: this refers to the time taken for the

MVM to be calculated on the Xeon Phi (or Xeon Phis) excluding any transfer times. $T_{CALC} = t_3 - t_2$ (see Fig 7.7).

- T_{TRANS} is the combined transfer time: this represents the time taken for the data to be transferred in and out of the Xeon Phi. This encapsulated both the transfer of the slope vector and DM commands. $T_{TRANS} = T_{OFFL} - T_{CALC}$ (see Fig 7.7).

- T_{RTCS} is the real-time control time: this represents the time taken for an entire AO frame to execute. This includes the WFS pixel processing, slope computation, the MVM calculation on the Xeon Phi(s) as well as any additional background tasks of the RTC.

At the start of the process, we initialise and pre-load the control matrix G^{-1} into the Xeon Phi memory. The slope vector is then transferred to the Xeon Phi (marked as time t_1) and the MVM calculation starts (time t_2). At the end of the calculation (time t_3) we transfer the result back to the host computer (time t_4) and loop back to the slope transfer. After a statistically significant number of timings (typically 10^6), we free the memory and end the process. Timings t_1 and t_4 are taken on the host computer whereas t_2 and t_3 are taken on the Xeon Phi. This timing scheme allows us to time the overall offload time T_{OFFL} and MVM calculation time T_{CALC} separately. From measuring T_{OFFL} and T_{CALC} the combined transfer time T_{TRANS} can be derived. Separating between transfer and calculation times offers us the capability to locate where additional time delays are being generated.

7.3.3 Multiple Xeon Phis

We have developed C code using pthreads to control multiple Xeon Phis simultaneously. This allows us to distribute and control multiple Xeon

Phis separately. In this study, we have tested the use of two identical Xeon Phis. We set the affinity of each thread so that they can run on each CPU as well as on separate cores on the same CPU. We found that it made no difference in performance. The main effect on performance was due to raising the priority of these threads so that they pre-empt the RT-OS.

Figure 7.8 shows the sequence diagram used for testing of the two Xeon Phis. The region in blue represents the host CPU. All the data is created on the main thread then distributed to the worker threads. Memory for the control matrix, wavefront slope vector and DM command vector are allocated on the Xeon Phis and the control matrix is uploaded in full to both cards. Once the memory has been allocated and uploaded to the Xeon Phis, the worker threads signal the main thread they are ready for the main loop.

The main loop starts on the main thread where the first timing (t1) is taken. The main thread broadcasts to all threads that the slope vector has arrived. Once the threads receive the command, they upload the entire wavefront vector to both Xeon Phis. Each of the Xeon Phis, once the vector has been uploaded, begin their section of the MVM. One Xeon Phi performs the first half and the other performs the second half. This method has the benefit of not needing a vector addition on the host after the computation. Alternatively, it is possible to upload the first half of the wavefront vector to the first Xeon Phi, and begin calculations while data is still being transferred to the host. Both of these methods were tested but for our purpose the chosen method was quicker.

Once the MVM calculation is finished, the first half of the DM com-

mand vector is downloaded from the first Xeon Phi and the second half from the second Xeon Phi. Once each half of the vector is received by its controlling thread, the thread signals the main thread. Once the main thread has received both signals it takes a second timing (i.e. t4).

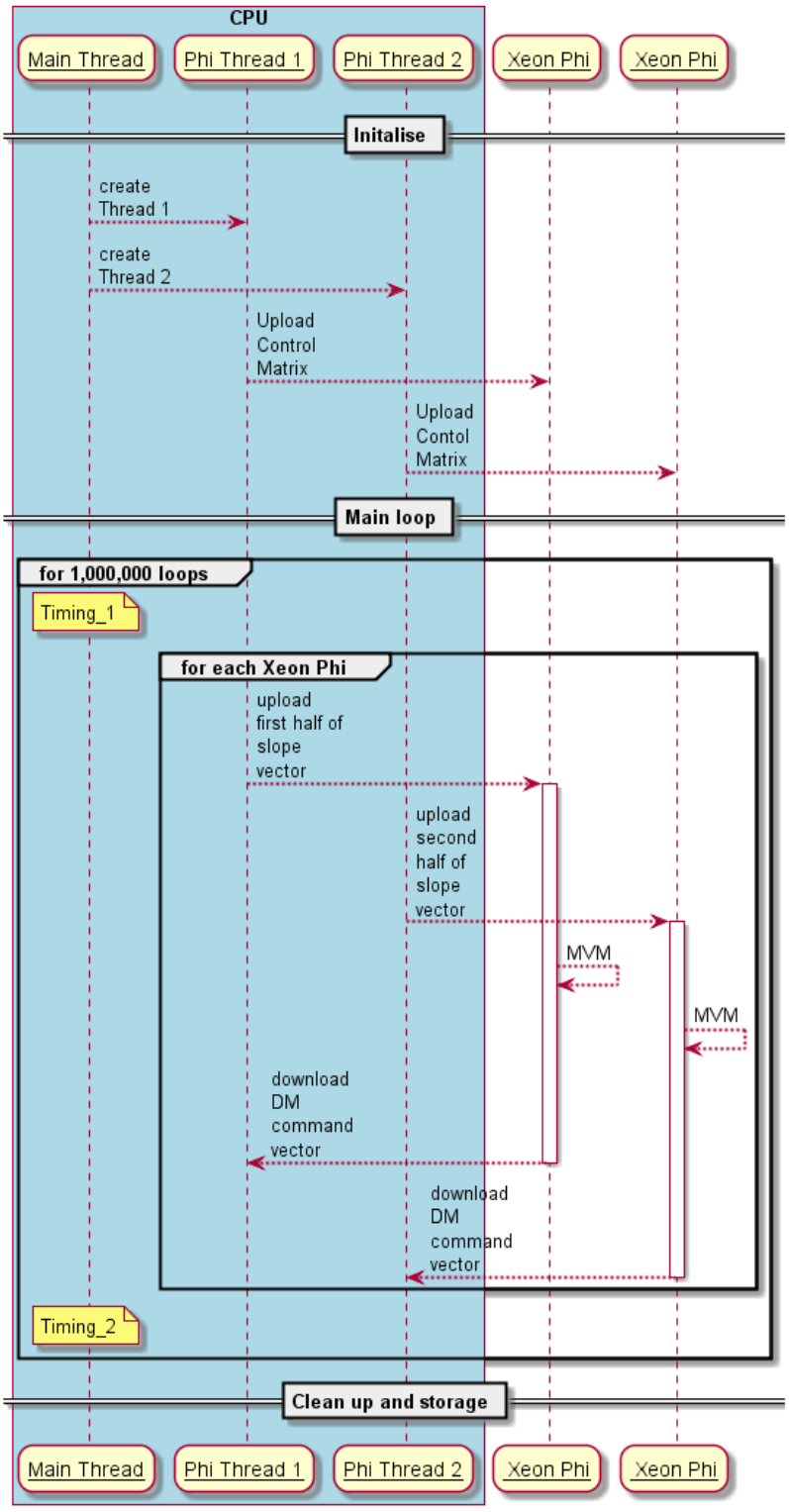


Figure 7.8: Sequence diagram for dual Xeon Phi setup.

7.3.4 Influence of system size

In order to stay as general as possible, and not tie this study to any specific instrument design, the AO system size (M) will be defined as the total number of wavefront measurement points (typically the number of valid sub-apertures of a SH-WFS). We will investigate the performance of the Xeon Phi for a range of input AO sizes. The slope vector size is 2M as it contains the slopes in X and Y directions for each valid sub-aperture. In a typical single conjugate AO (SCAO) system, the number of valid sub-apertures is approximately equal to the number of DM actuators and so the DM command vector will be approximately half the size of the slope vector. The control matrix (G^{-1}) will therefore be of dimension M×2M unless explicitly stated otherwise.

7.3.4.1 Optimisation of the problem size to the architecture

Figure 7.9 shows a periodic pattern of the mean offload time that is observed as we increase the problem size. This suggests that the Xeon Phi performs best when the problem size can fit its architecture. The Xeon Phi 5110p has 60 cores, with one core reserved for input/output routines. Each core can support up to four threads giving a total of 236 supported threads (4×59). When the problem size is divisible by 236 optimal performance is obtained. This difference in performance is likely due to the architecture and also how the MKL library handles the parallel sections of code. When the problem size fits the architecture, the library is able to split the problem evenly between all cores. When this is not the case, the library has to perform some dynamic resource handling that brings in more overhead, degrading performance. When the problem size is not divisible by 236, the control matrix is therefore

padded with zeros to fit the architecture and reach the best achievable performance.

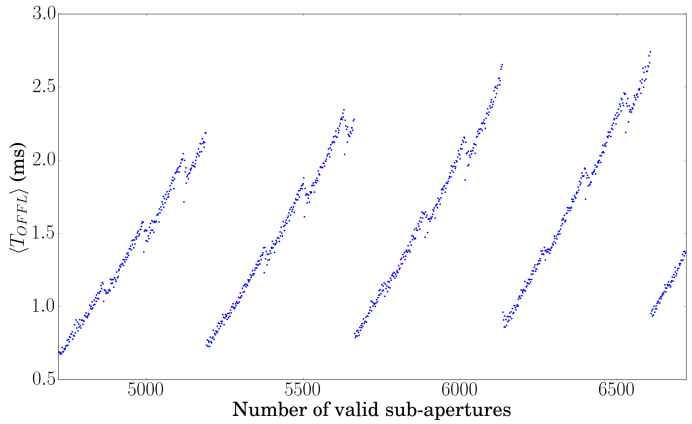


Figure 7.9: Comparison of the mean offload time $\langle T_{OFFL} \rangle$ as a function of the number M of valid sub-apertures. Measurements are averaged over 10^4 samples.

When using two Xeon Phis, we have twice the number of cores and threads. This means the MVM is split across 118 cores (2×59) and 472 threads (118×4). Figure 7.10 shows that when the wavefront slope vector is a multiple of 436 we get the best performance. Due to this effect, for a dual Xeon Phi configuration we are padding the matrix similarly to the single Xeon Phi to a multiple of 472.

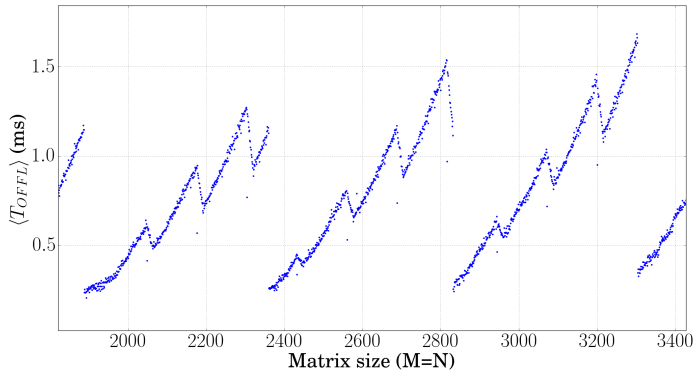


Figure 7.10: Comparison of the mean offload time $\langle T_{OFFL} \rangle$ using 2 Xeon Phis, as a function of the number M of valid sub-apertures. Measurements are averaged over 10^4 samples.

7.3.4.2 Offload time as a function of AO system size

In figures 7.11 and 7.12 (a zoom of 7.11 for smaller systems sizes) we present a comparison of mean offload time $\langle T_{OFFL} \rangle$ (i.e. including MVM calculation and data transfer) for the dual Xeon E5 CPU (red), a single Xeon Phi (blue) and 2 Xeon Phis (green) used as accelerators as a function of AO size.

For the smaller AO systems where the number of valid sub-apertures is less than approximately $M < 1500$ (such as for a 40×40 SCAO system) the Xeon E5 clearly outperforms the Xeon Phi(s). This is due to the need to transfer data (i.e. slope and DM command vectors) over the PCIe bus. Once the number of valid sub-apertures becomes larger, the Xeon Phis provide lower mean latencies.

As expected, for large numbers of valid sub-apertures $\langle T_{OFFL} \rangle$ grows as $\mathcal{O}(M^2)$ for all devices, dominated by computation time. When two Xeon Phi(s) are used, $\langle T_{OFFL} \rangle$ can be further reduced; the difference is more clearly visible for large AO systems. For example, this allows us to perform the MVM for an 80×80 system in 1.2 ms using 2 Xeon Phis (resp. 1.8 ms for 1 Xeon Phi and 6 ms with the Xeon E5).

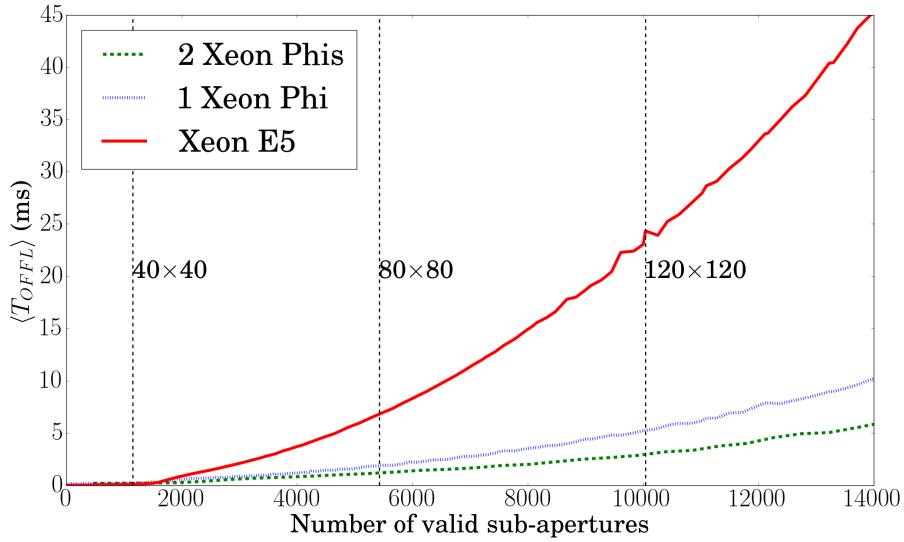


Figure 7.11: Comparison of the mean offload time $\langle T_{OFFL} \rangle$ as a function of the number M of valid sub-apertures using 10^4 samples. Red: Xeon E5; Blue: single Xeon Phi; Green: two Xeon Phis. Results obtained using a RT Linux kernel. The dashed vertical lines represent the approximate size of an AO system in total number of sub-apertures.

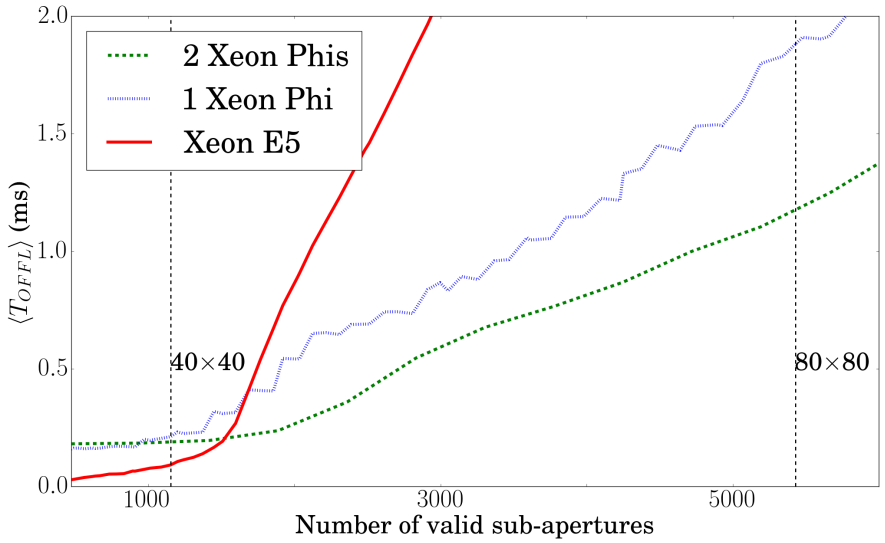


Figure 7.12: Comparison of the mean offload time $\langle T_{OFFL} \rangle$ as a function of the number M of valid sub-apertures using 10^4 samples. This figure is a zoom of figure 7.11 for better readability.

As stated previously, the mean execution time is insufficient to fully characterise the performance of AO RTC system or sub-systems. Fig-

ure 7.13 shows the distribution of offload times T_{OFFL} as a function of AO system size, using a RT-Linux kernel, and measuring T_{OFFL} over 10^6 iterations for each system size. Only matrix size multiples of 236 are shown because mean and variation in execution time are both increased for non-multiples. It can be seen that not only all system sizes have a similar bimodal distribution but the mean, the position of peaks and the different percentiles calculated (percentile of offload times that are completed by the given time; e.g. 99% of the calculation will take less than P99% to execute) all follow a similar trend. Only the maximum offload time is shown to be somewhat irregular; we believe this is because of the limited number of samples used (i.e. 10^6) to calculate the distribution. For example, an 80×80 system will finish the MVM calculations 99.99% of the time before approximately 3 ms.

From figure 7.13, we can legitimately say that studying a specific AO system size in detail will not remove the generality of the analysis as results can be scaled to match the desired system size. Results obtained with 2 Xeon Phis (data not shown here) show the same scalability.

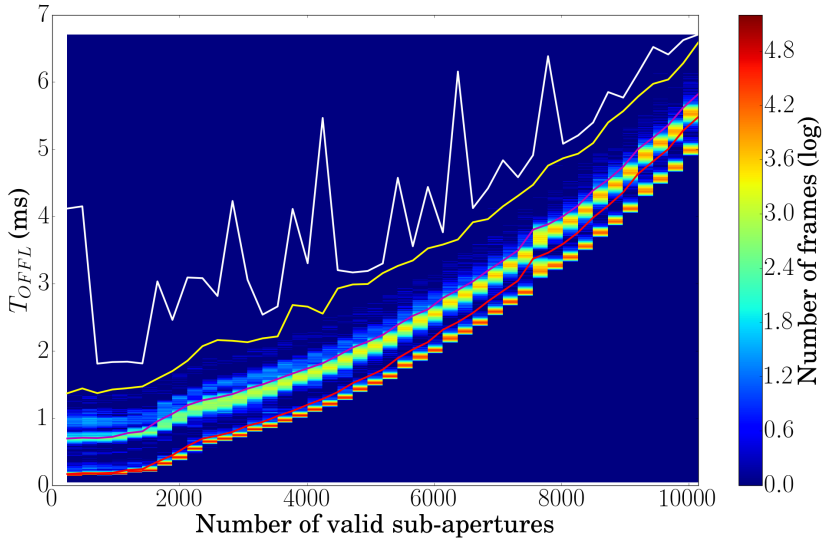


Figure 7.13: Variation in offload time T_{OFFL} using a single Xeon Phi with the host running a RT Linux Kernel. Only multiples of 236 are shown as they provide best performance. A logarithmic colour lookup table is use to visualise both peaks and tails of the distributions. The red line represents the mean, purple represents $P_{99\%}$, yellow represents $P_{99.99\%}$ and the white line represents the maximum time measured.

7.3.4.3 Influence of shape of the control matrix

So far we have discussed the performance of the Xeon Phi in the context of SCAO systems, where the number of DM actuators is approximately equal to the number of sub-apertures and where we have assumed that the control matrix shape is $M \times 2M$. In this section, we investigate how the shape of the control matrix affects the general performance of the MVM calculation.

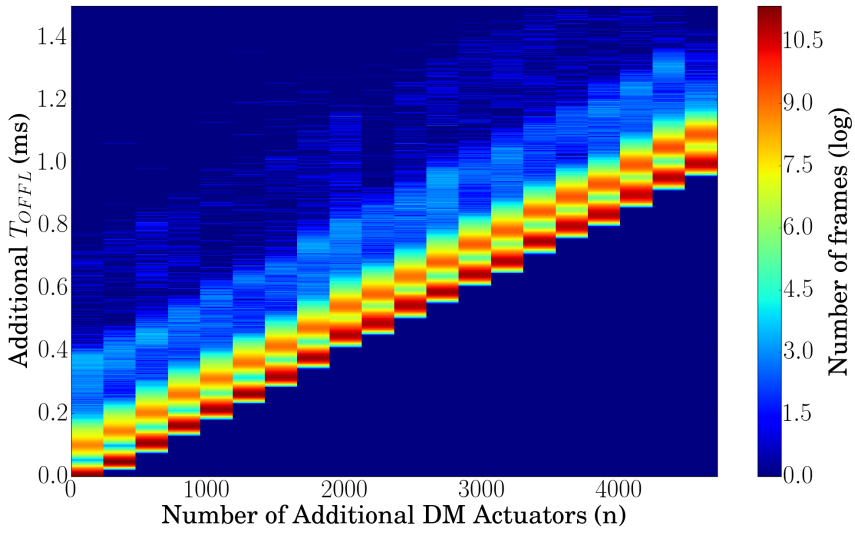


Figure 7.14: Increase of offload time T_{OFFL} as the number of elements of the control matrix is increased from $M \times 2M$ to a square with $2M \times 2M$ elements (the control matrix has $(M + n) \times (2M)$ elements, where $0 \leq n \leq M$ and $2M = 9440$). Data calculated using 10^6 samples. A logarithmic colour lookup table is use to visualise both peaks and tails of the distributions.

Figure 7.14 shows the additional offload time T_{OFFL} taken as we increase the control matrix size from $M \times 2M$ to a square with $2M \times 2M$ elements (i.e. the control matrix has $(M + n) \times (2M)$ elements, where $0 \leq n \leq M$). The baseline system (i.e. where $n = 0$) is equivalent to an 80×80 AO system with $2M = 9440$. As we have stated previously, all dimensions of the matrix are a multiple of 236 to maximise performance (explaining the steps in performance for every increase of n by 236). We see that the time increase is linear (i.e. increases linearly with the number of additional control matrix elements) and the overall shape of the distribution remains identical for all system sizes (double peak with a tail). It is important to stress at this point that using figure 7.13 in conjunction with figure 7.14 enables us to estimate the performance of a single Xeon Phi for most AO system sizes, regardless of the size or shape of its control matrix.

7.3.4.4 Multiple Xeon Phi speedup

An MVM operation is highly parallelisable, and therefore easy to split between multiple Xeon Phi accelerators. Adding a second Xeon Phi doubles the available compute power and memory bandwidth. However, synchronisation between the two processes makes achieving a speedup of $\times 2$ difficult.

Figure 7.15 shows the offload time speedup that can be achieved by using 2 Xeon Phis instead of one. For small systems using 2 Xeon Phis is actually slower than just relying on one, due to overheads introduced. As the control matrix size increases, the speedup gradually rises to reach a plateau (starting from about a 120×120 AO system) of approximately 1.8. Using a second Xeon Phi to complete the MVM allows us to have twice the cache memory (60 MB). This explains the first performance peak on figure 7.15 while a single Xeon Phi would have to access the slower GDDR5 memory, the two Xeon Phis are able to fit the control matrix within the available combined cache memory. A single computer can contain up to 8 Xeon Phis, as long as it has a sufficient number of PCIe bus lanes.

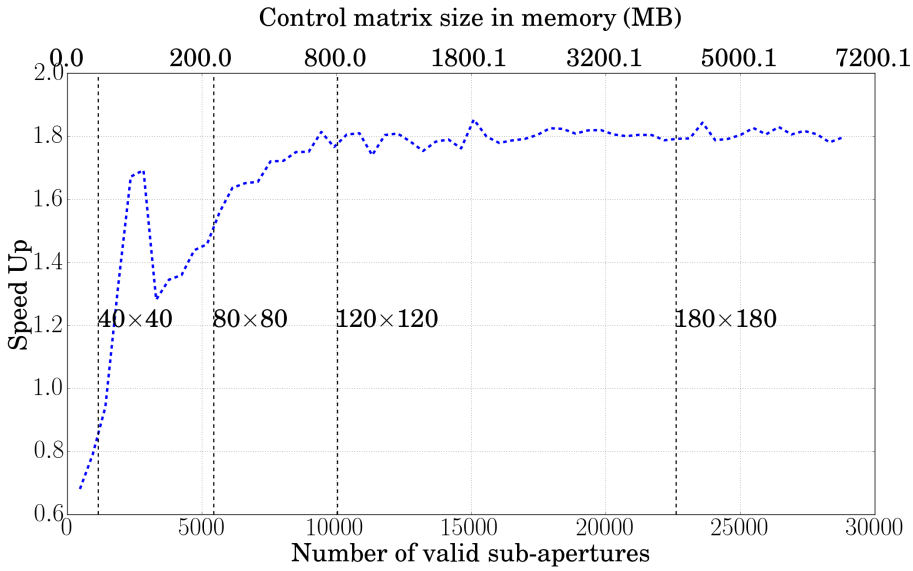


Figure 7.15: Relative performance of using two Xeon Phis instead of one: $\langle T_{OFFL} \rangle^{2Phis} / \langle T_{OFFL} \rangle^{1Phi}$ as a function of system size. The dashed vertical lines represent the approximate size of an AO system in total number of sub-apertures.

7.3.5 Detailed analysis of temporal behaviour

As we have seen in the previous section, mean offload time does not enable us to fully characterise where the different latencies are coming from and how they affect the calculation speed. Access to the full distribution of computation times is therefore necessary. In addition, we have checked that studying a specific AO system size will not remove the generality of the analysis as results can be scaled to match the desired system size. In this section, we analyse detailed results for a typical E-ELT first-light AO instrument with 80×80 sub-apertures (using a 9440 × 5428 element control matrix, or 205 MB).

7.3.5.1 Variability in offload time: T_{OFFL}

Figure 7.16 shows the distribution of T_{OFFL} for three tested configurations (i.e. 1 Xeon Phi used with a non-RT Linux host, and 1 and 2 Xeon Phi used with a RT Linux host). For each configuration, 10^6

measurements were taken. Table 7.6 shows more details on the offload time, analysing the distributions in terms of minimum and maximum values, jitter, mean and percentiles. The percentiles $P_{XX\%}$ are defined as the time by which $XX\%$ of the samples are completed. In other words $P_{99.99\%}$ represents a 1 in 10,000 event. Also shown in table 7.6 are the results of the Xeon E5 completing the same MVM calculation on a RT Linux system.

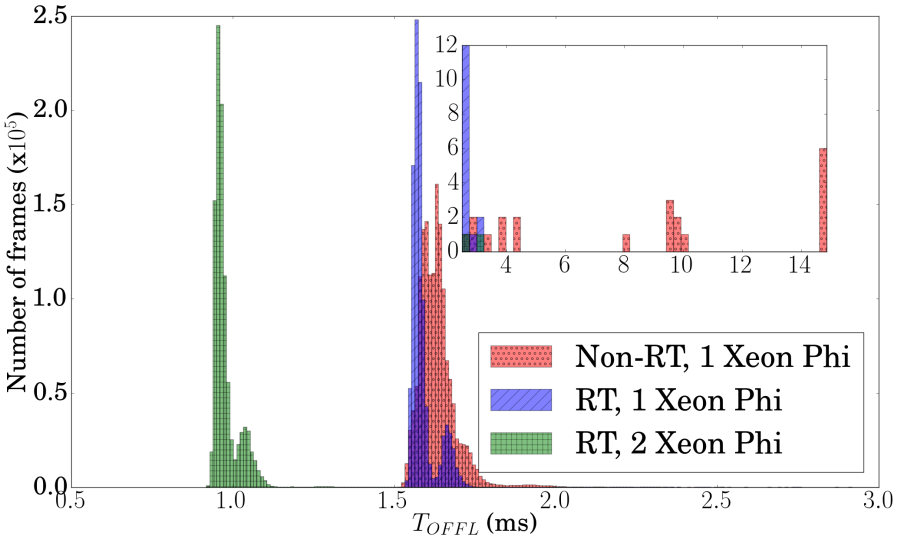


Figure 7.16: Histogram comparing the offload time T_{OFFL} for a 80×80 sub-aperture system calculated using 10^6 samples. T_{OFFL} encapsulates both calculation time and transfer time. Blue: Single Xeon Phi on a RT Linux host; Red: single Xeon Phi on non-RT Linux; Green: Two Xeon Phi on RT Linux kernel. Inset shows data from 2.5 ms to 15 ms with the number of frames ranging from 0-12 (showing outliers more clearly).

All three distributions are double peaked with a long tail due to outliers;[14] have published a similar distribution. The shape of these distributions causes the mean not to sit at $P_{50\%}$ (the median value) but at $P_{56\%}$ for non-RT Linux and at $P_{73\%}$ for RT. Using a RT Linux does not appear to greatly decrease the mean latency when compared to the non-RT Linux. It seems however to reduce the majority of the

extreme outliers and significantly lower the jitter (which is defined as the deviation σ) from 0.066 ms to 0.039 ms. As expected, two Xeon Phis on a RT Linux system produces the lowest latency; $\langle T_{OFFL} \rangle$ is about 1.6 times less than on a single Xeon Phi. However, jitter is increased. This is not entirely surprising as to complete the MVM both Xeon Phis need to have finished their calculation. This means that for a given frame, jitter is introduced by the worst-performing Xeon Phi. The outliers occur so infrequently that $\langle T_{OFFL} \rangle$ is unaffected.

Table 7.6: Offload times T_{OFFL} corresponding to figure 7.16. $P_{N\%}$ is the N^{th} percentile of offload times that are completed by the given time. All times given in milliseconds.

T_{OFFL} (ms)	1 Xeon Phi non-RT	1 Xeon Phi RT	2 Xeon Phis RT	Xeon E5 RT
Jitter (σ)	0.066	0.039	0.0426	0.057
Min	1.514	1.525	0.918	3.480
Mean ($P_{XX\%}$)	1.631 ($P_{56\%}$)	1.587 ($P_{73\%}$)	0.978 ($P_{71\%}$)	3.622 ($P_{60\%}$)
$P_{99\%}$	1.863	1.704	1.097	3.661
$P_{99.9\%}$	2.009	1.765	1.320	3.865
$P_{99.99\%}$	2.099	2.198	1.678	4.035
$P_{99.999\%}$	4.295	2.663	2.118	15.394
Max	14.861	3.085	3.119	32.170

Even by using two Xeon Phis, the number of outliers measured may still be a concern for certain AO configurations. With 1 outlier every 10000 frames (i.e. $P_{99.99\%}$) for a system running at 500 Hz will occur 180 times over the course of an hour of observation. To identify where these outliers are arising from and see if they can be reduced, we investigate in the following sections the split of T_{OFFL} into its component T_{CALC} and T_{TRANS}.

7.3.5.2 Variability in calculation time: T_{CALC}

Since the MVM is only calculated on the Xeon Phi, it does not directly interact with the operating system running on the host computer. We do not expect to see any changes in the distribution of the MVM calculation time T_{CALC} on RT or non-RT systems[4]. However compiling the Xeon Phi drivers to be compatible with a RT Linux meant upgrading the Manycore Platform Software Stack (MPSS) to the latest version at the time of writing[5]. This update brought updated versions of Flash and System Management Controller (SMC). MPSS and the updates are only partially open source. This suggests that issues arising from updates may be solvable by editing these sections of the source code without the manufacture's support. However, some sections are closed-source which may make user modifications more difficult.

Figure 7.17 shows the results for pre-updated Flash/SMC and after the update was applied. It seems to suggest that the update caused a reduction in performance and larger variations in timings. The variation in calculation time T_{CALC} is probably due to the fact that the Xeon Phi is running a non-RT micro-Linux which results in some large outliers[3]. In table 7.7 we see that the update slightly reduces both mean calculation times and jitter. It is not until $P_{99.99\%}$ that we see that the outliers become worse after the update. This is a problem for the performance of the system, it also highlights a larger problem of using hardware accelerators such as the Xeon Phi or GPUs. Neither of these technologies are designed for RTC, and any AO RTC system based on hardware accelerators is tied to the development and direction

[4]This was shown to be true by going back to a non-RT system after update.

[5]MPSS 3.4 (Flash 390-2; SMC: 1.16) from Linux Gold Update 3 (Build: 2.1.6720-13; Flash:386-2; SMC:1.14))

the company developing them decides on. An upgrade that is beneficial to HPC or gaming community may degrade AO RTC performance. As a result, and since mean and jitter cannot fully characterise hardware for AO applications, it is necessary to analyse the full distribution of frame computation times when comparing or upgrading hardware.

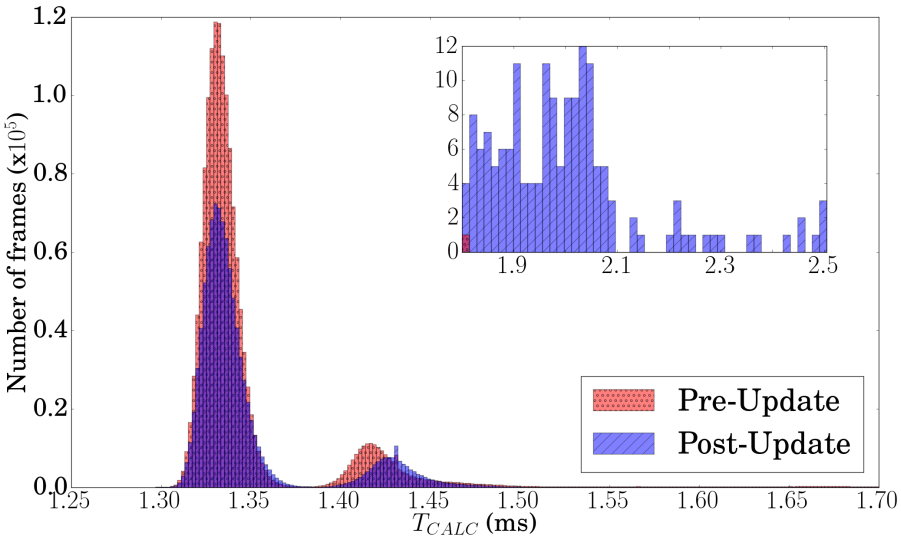


Figure 7.17: Histogram comparing the calculation time T_{CALC} for a 80×80 sub-aperture system calculated using 10^6 samples. T_{CALC} excludes the transfer time. Blue: single Xeon Phi post firmware update; Red: single Xeon Phi pre firmware update. Inset shows data from 1.8 ms to 2.5 ms with the number of frames ranging from 0-12 (showing outliers more clearly).

Table 7.7: Calculation times T_{CALC} corresponding to figure 7.17. $P_{N\%}$ is the N^{th} percentile of MVM calculation times that are completed by the given time. All times given in milliseconds.

T_{CALC} (ms)	1 Xeon Phi Pre-update	1 Xeon Phi Post-update
Jitter (σ)	0.045	0.037
Min	1.297	1.301
Mean	1.349 ($P_{81\%}$)	1.348 ($P_{77\%}$)
$P_{99\%}$	1.560	1.463
$P_{99.9\%}$	1.701	1.522
$P_{99.99\%}$	1.749	1.957
$P_{99.999\%}$	1.765	2.306
Max	1.813	2.505

7.3.5.3 Variability in data transfer: T_{TRANS}

The larger outliers seen in T_{OFFL} were not seen in T_{CALC}, suggesting that the main cause lies in transfer time T_{TRANS}. Figure 7.18 shows the distribution of data transfer timings for a host computer running both a non-RT and a RT Linux system. Table 7.8 shows the detailed results for the data transfer. It demonstrates that the large outliers seen in the non-RT T_{OFFL} are indeed caused by the transfer of data between the host computer and the accelerator. We see that moving to a RT Linux has reduced the outliers bringing the maximum values from 13.507 ms down to 1.747 ms, a large reduction. It has also suppressed most of the outliers, but not all, and reduced jitter from 0.048 ms to 0.014 ms. On average, the system spends 15% of the total offload time transferring data in and out of the Xeon Phi. When adding a second Xeon Phi (data not shown here), the transfer time T_{TRANS} is not reduced by much, even though only half the data needs to be transferred to each Xeon

Phi. This transfer happens simultaneously but due to the overheads involved, T_{TRANS} cannot be reduced by a significant amount.

In order to increase robustness, a method for reducing transfer (and calculation) time variability could be devised by using two Xeon Phis performing identical calculations. When the fastest Xeon Phi has finished its calculations, the other is interrupted to be ready to receive slopes from the next frame. There is no simple functionality to interrupt a call to the MKL library running on the Xeon Phi once it has started or to stop the data transfer. This issue has not been investigated here, and it is believed that the next Xeon Phi generation (standalone CPU with no data transfer, see section 7.4) will be able to run a RT Linux system therefore removing almost entirely these very high latency events.

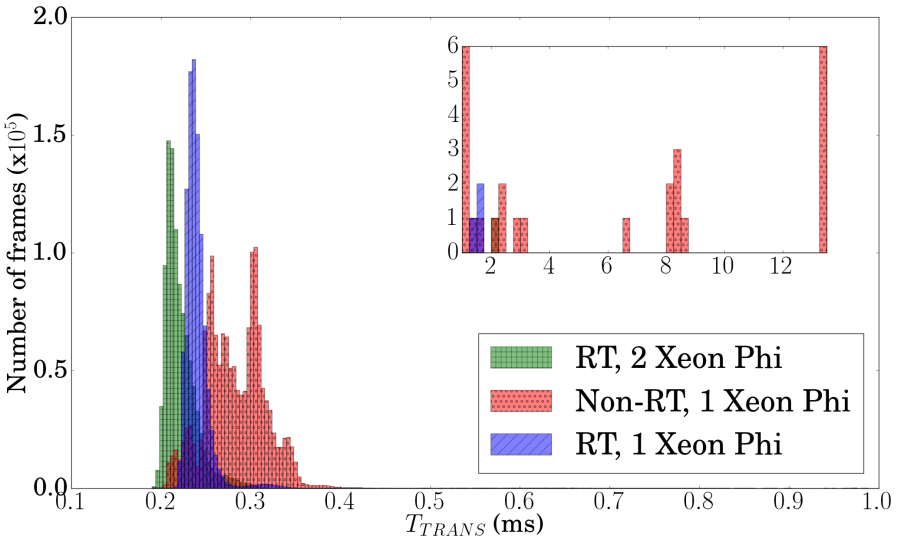


Figure 7.18: Histogram comparing the transfer time T_{TRANS} for a 80×80 sub-aperture system calculated using 10^6 samples. T_{TRANS} is the combined time for transferring data in and out of the Xeon Phi. Blue: Single Xeon Phi on a RT Linux kernel; Red: single Xeon Phi on non-RT Linux. Inset shows data from 1 ms to 13 ms with the number of frames ranging from 0-6 (showing outliers more clearly).

Table 7.8: Transfer times T_{TRANS} corresponding to figure 7.18. $P_{N\%}$ is the N^{th} percentile of transfer times that are completed by the given time. All times given in milliseconds.

T_{TRANS} (ms)	1 Xeon Phi Non-RT	1 Xeon Phi RT
Jitter (σ)	0.048	0.014
Min	0.196	0.204
Mean	0.283 ($P_{81\%}$)	0.239 ($P_{77\%}$)
$P_{99\%}$	0.359	0.313
$P_{99.9\%}$	0.592	0.344
$P_{99.99\%}$	0.752	0.399
$P_{99.999\%}$	2.957	0.647
Max	13.507	1.747

7.3.5.4 Integration of Xeon Phi into an AO RTC software: T_{RTCS}

We have shown that the Xeon Phi is able to reduce MVM calculation time for large systems over that of modern CPUs, such as the Xeon E5. In this section, we integrate the Xeon Phi into a complete AO RT control software. We chose to integrate the Xeon Phi into Durham Adaptive optics Real-time Controller (DARC)[21], which is currently being used on the William Herschel Telescope for the CANARY AO demonstrator. In this section, we run an end-to-end simulation of an AO RTC system using simulated WFS data. The measured time T_{RTCS} includes WFS pixel processing, slope calculation, the MVM calculation on the Xeon Phi as well as additional background tasks of the RTC system.

Only a single thread is able to transfer data to the Xeon Phi at one time. The transfer step has a larger amount of overhead when compared the data transfer size. This overhead means that if we pipeline the AO

control loop and transfer groups of slopes to the Xeon Phi and split the MVM into multiple smaller MVMs, the overall T_{RTCS} is increased. Although pipelining the MVM is possible, it was decided here to perform a single MVM per frame, when all slopes have been calculated. For the next generation of the Xeon Phi, where there is no transfer step, pipelining will be more appropriate.

Figure 7.19 shows the comparison between T_{RTCS} running both non-RT and RT Linux using a Xeon Phi to accelerate the MVM. Table 7.19 shows more detailed results of the RTC processing time, analysing the distributions in terms of minimum and maximum values, jitter, mean and percentiles. It also shows the results for the RTCS running on the Xeon E5 only, without Xeon Phi acceleration. Offloading the MVM to the Xeon Phi and running on a RT Linux system brings the jitter down (i.e. narrower distribution) and reduces the number of outliers, although they are not completely eliminated. The jitter of the whole RTC is reduced from 0.193 ms for a non-RT Linux down to 0.061 ms for a RT Linux, a sizable reduction. The outliers appear to be far worse than for the standalone tests; this is likely to be due to the fact that the CPU is now stressed with other tasks such as WFS data processing and thread synchronisation. As expected, moving to a RT kernel has made the maximum value drop, from 103 ms to 11.8 ms. The minimum and mean values have however increased slightly on the RT system; this is likely due to how the internals of the RT kernel work allowing a process with raised priority to pre-empt other processes. RT Linux systems do not optimise for overall performance, they optimise for reliability and predictability.

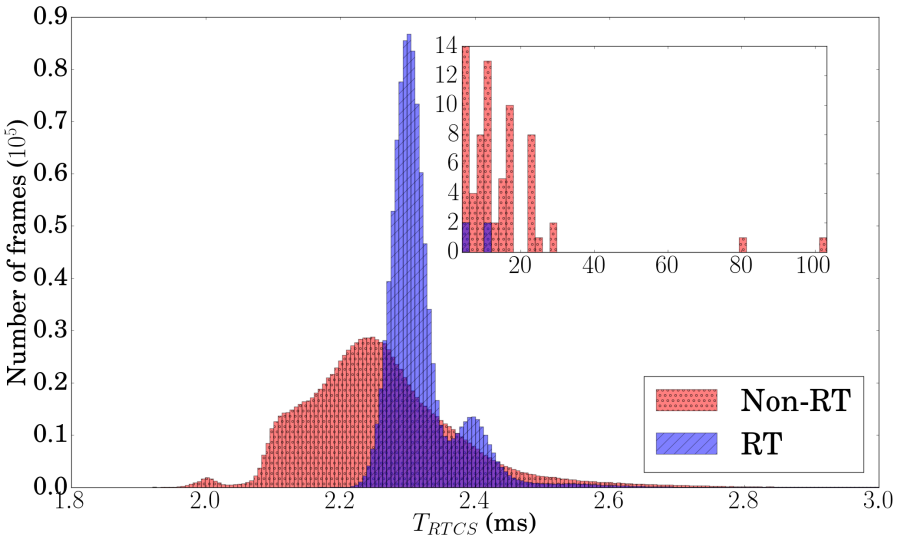


Figure 7.19: Histogram comparing the entire AO frame processing T_{RTCS} for a 80×80 sub-aperture system calculated using 10^6 samples. T_{RTCS} encapsulates MVM calculation time, transfer time, pixel processing, slope computation and any overhead of running the RTC system. Blue: single Xeon Phi on RT Linux kernel; Red: single Xeon Phi on a non-RT Linux. Inset shows data from 4 ms to 103 ms with number of frames ranging from 0-14 (showing outliers more clearly).

Table 7.9: Entire AO frame processing times T_{RTCS} corresponding to figure 7.19. $P_{N\%}$ is the N^{th} percentile of entire AO frame processing times that are completed by the given time. All times given in milliseconds.

T_{RTCS} (ms)	1 Xeon Phi non-RT	1 Xeon Phi RT	Xeon E5 non-RT	Xeon E5 RT
Jitter (σ)	0.193	0.061	1.5924	0.570
Min	1.920	2.154	6.120	6.717
Mean ($P_{XX\%}$)	2.260 ($P_{56\%}$)	2.320 ($P_{67\%}$)	8.550 ($P_{54\%}$)	8.366 ($P_{53\%}$)
$P_{99\%}$	2.642	2.545	11.058	10.065
$P_{99.9\%}$	2.848	2.921	11.455	10.816
$P_{99.99\%}$	3.179	3.267	16.345	11.362
$P_{99.999\%}$	22.504	3.570	20.727	11.773
Max	103.249	11.759	128.028	12.077

Although this performance would not be sufficient for typical first-light E-ELT instruments (e.g. mean frame time of 2.3 ms), we have demonstrated in this section that incorporating the Xeon Phi into an existing AO RT control software can be done efficiently without investing a significant amount of time and effort and has the potential to improve the RTC performance.

7.4 Prospective evolution of the Xeon Phi

The Xeon Phi tested in this chapter is the Xeon Phi 5110p and was released as a coprocessor for use as a hardware accelerator. The next generation of the Xeon Phi, codenamed 'Knights Landing'[22], has been released in the middle of 2016. The specifications for the top-end chip can be found in table 7.10. This chip is no longer a coprocessor but a CPU that is directly plugged into the motherboard.

The next generation Xeon Phi cores are based on Intel Atom CPUs which have a 512 kB of cache of each and a total number of cores upwards of 60, depending on the model. The clock frequency has been increased to ≈ 1.5 GHz, which is still slower than modern CPUs, making it suffer the same performance problem when running serial code. Each core has 512 KB of L2 cache so the Knights Landing will have around 35 MB of L2 cache, the exact value depending on the model.

The size of control matrix for ELT first light instruments is however considerably larger than this (at least 205 MB) and we can safely assume that the MVM will still be memory bandwidth limited. The Knights Landing has removed the GDDR5 memory and is now using DDR4 memory which is much slower, though the use of this standard memory type allows for expansion of the memory. The 7290F supports up to

384 GB of memory with a memory bandwidth of 115.2 GB s^{-1}[22].

This is much slower than the previous generation and will cause issues with the MVM. This is why Intel has added a mid point memory between DDR4 and the L2 cache: the MCDRAM which is 4 times faster than DDR4. The Knights Landing has up to 16 GB of this memory integrated into the chip. This MCDRAM allows it to approach an effective memory bandwidth close to 500 GB s^{-1}(according the STREAM tests performed by Intel).

From these specifications we can use the average performance (see figure 7.20) of the current Xeon Phi and predict the performance of the next generation as well as NVIDIA competing cards the K80 and P100[23],[24]. This shows us how the performance might scale as a function of system size. The mean performance of the Knights Landing rivals that of two Xeon Phis of current generation.

The large increase in performance is mainly due to the removal of the data transfer step and to higher memory bandwidth. Jitter and outliers are harder to predict. It is reasonable however, to assume that the distribution curve for T_{CALC} will be similar to that of the current generation and that running a RT-Linux kernel on the Xeon Phi will further reduce both jitter and the number of outliers. The next Xeon Phi generation, being a standalone CPU, has the potential to offer the performance benefits of the current hardware accelerators (e.g. Xeon Phi, GPUs) while removing the main disadvantages of this technology: the transfer of data.

From our models we predict that the Knights Landing outperforms even the P100. The P100 has a higher memory bandwidth but still requires data to be offloaded.

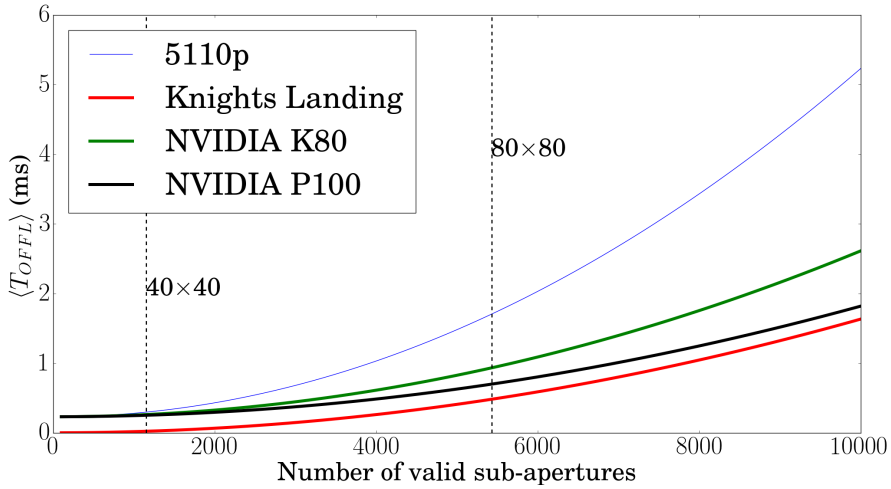


Figure 7.20: Comparison of mean MVM execution time between the current Xeon Phi (5110P) (blue), the predicted next generation (red), NVIDIA K40 (green) and NIVIDA P100 (black). These are predictions based on the data sheets available for each of these chips[23, 24].

In this investigation we have only considered the use of a unique control matrix throughout the operation of the AO system. In reality, the control matrix will need to be updated as the observation condition changes; for the E-ELT this is likely to be of the order of minutes. The Xeon Phi offers asynchronous transfer of data which should allow for the matrix to be updated during calculations. Since the MVM is a memory bandwidth limit problem, and that data transfers will require access to memory, this is likely to reduce performance.

The next generation of Xeon Phi (standalone CPU) should allow control matrix swapping without transferring data over the PCIe bus, therefore mitigating its impact. Transferring an updated control matrix into memory will still reduce the memory bandwidth for the MVM calculation and reduce performance. To lessen the impact on performance, the updated matrix could be uploaded over several iterations, updating small sections at a time.

Table 7.10: Specifications of Knights Landing 7290F, one of the next generation Xeon Phi which are now being released.

Processor Number	7290F
Cores / Threads	72/288
Clock frequency	1.5 GHz
L2 cache	36 MB
MCDRAM	16 GB
DDR4 RAM	up to 384 GB

While the Xeon Phi targets MCDRAM to improves the memory bandwidth, many competing hardware technologies are moving towards 3D stacked memory sometimes referred to as High Bandwidth Memory. This new type of memory allows for a wider memory bus (1024 bits wide), higher memory bandwidths and lower operating powers. Nvidia are now releasing GPUs with either GDDR5 or HBM[25]. At the current time the difference in performance between MCDRAM and HBM is minor, in the future this may grow. HBM offers an interesting prospect for future systems and will need further investigation.

7.5 Conclusions

In this chapter, we have presented a detailed study of the Xeon Phi, a many-core processor, used as hardware accelerator for AO real-time applications. We investigated performance for the most compute intense part of the RTC: the wavefront reconstruction. Our examination focused on the MVM algorithm, the most commonly used and the most parallelisable of wavefront reconstruction algorithms. This enables us to take full advantage of the high number of cores of the Xeon Phi. We described how AO system size and the number of Xeon Phis im-

pact performance and investigated the main contributors to the time delay splitting between data transfer time and MVM calculation. Finally, we discussed the implementation ease and overall performance of integrating the Xeon Phi into a complete RTC software.

We demonstrated that performance changes gradually over the whole range of control matrix sizes studied, and that performance for a specific AO system can easily be assessed by scaling. We believe that the results obtained here can serve as a guideline for estimating MVM performance for any AO system size using a single or multiple Xeon Phis. A more detailed analysis also showed that mean execution time is rarely sufficient to fully qualify novel hardware (or when updating firmware) in RT applications and that the actual distribution of execution times needs to be analysed in detail. To make the comparison between potential RTC hardware more tractable, we decided to present results in terms of minimum, maximum, mean, deviation (jitter) and percentile of execution time.

Using the Xeon Phi enables a clear improvement in MVM mean calculation time, whether tested as a standalone system or fully integrated into the RTC software. We have shown that moving the host from a non-RT to a RT Linux system can naturally reduce the number and extent of outliers, as well as reduce mean offload times. For a typical 80×80 E-ELT first-light SCAO system, mean offload time $\langle T_{OFFL} \rangle \approx$ 1.587 ms and 99.999% of the offloads are finished within ≈2.663 ms. However, a number of outliers are still present (most likely due to the fact that the Xeon Phi is running a non-RT micro-Linux) probably making the current generation of this technology only suitable (TBC) for some AO RT applications (e.g. GLAO, MOAO) but unsuitable for

others (e.g. XAO).

Sharing calculations between two Xeon Phis allows us to further reduce mean offload time $\langle T_{OFFL} \rangle$. The maximum speedup between 1 and 2 Xeon Phis plateaus at around 1.8 for large systems, and the speedup for a typical 80×80 E-ELT first-light SCAO system reaches 1.6. In this configuration, the mean offload time $\langle T_{OFFL} \rangle \approx 0.978$ ms and 99.999% of the offloads are finished within ≈2.118 ms. This shows the scalability of a system using multiple Xeon Phis, and it is reasonable to assume that adding more Xeon Phis would further reduce the latency in a similar way.

The Xeon Phi is designed to be used within supercomputers, Tianhe-2 powered by the Xeon Phi has spent 3 years at the top of the top500 list, only being beaten in the June 2016 list. The HPC community is generally more focused on data throughput rather than on time-critical processes. We have found that the variability in execution time (increased jitter and outliers) can increase after firmware updates. Using the Xeon Phi as an offload card turns a homogeneous CPU system into a heterogeneous computing environment, which is more complex to programme and to balance work loads efficiently. On the other hand, the theoretical memory bandwidth of the Xeon Phi is very high, which is essential for a bandwidth limited problem such as the MVM. We have shown that about 50% of the theoretical memory bandwidth is achievable, in line with other findings [16]. In addition, we have shown that the achievable memory bandwidth can offer a good estimate for the mean performance of the Xeon Phi calculating the MVM, and that most of the outliers come from transferring data in and out of the Xeon Phi. The expected next Xeon Phi generation has great potential in

being suitable for AO, being an integrated CPU, eliminating the need to transfer data over the PCIe bus, and also offering higher compute power. Both mean RTC performance, jitter and outliers have the potential to be greatly reduced in forthcoming hardware.

Due to the performance of the current Xeon Phi and the predicted performance of Knights Landing, future AO RTCs are already being designed around the Xeon Phi chip. A first light instrument on the Thirty Meter Telescope, the Narrow Field Infra-Red AO System (NFIRAOS), is now planning on using the Knights Landing at the centre of the RTC[26].

References

[1] Lisa Poyneer, Don Gavel, and James Brase. Fast wave-front reconstruction in large adaptive otpics systems with use of the fourier transform. *J. Opt. Soc. Am. A*, 19(10):2100–2111, 2002.

[2] Matthias Rosensteiner. Wavefront reconstruction for extremely large telescopes via cure with domain decomposition. *J. Opt. Soc. Am. A*, 29(11):2328–2336, 2012.

[3] Jean-Pierre Véran, Corinne Boyer, Brent L. Ellerbroek, Luc Gilles, Glen Herriot, Daniel A. Kerley, Zoran Ljusic, Eric A. McVeigh, Robert Prior, Malcolm Smith, and Lianqi Wang. Results of the nfiraos rtc trade study. In *Proc. SPIE*, volume 9148, page 91482F, 2014.

[4] Damien Gratadour, Arnaud Sevin, Julien Brulé, Eric Gendron, and Gérard Rousset. Gpus for adaptive optics: simulations and real-time control. In *Proc. SPIE*, volume 8447, page 84475R, 2012.

[5] A. Sevin, D. Perret, D. Gratadour, M. Lainé, J. Brulé, and B. Le Ruyet. Enabling technologies for gpu driven adaptive optics real-time control. In *Proc. SPIE*, volume 9148, page 91482G, 2014.

[6] Alastair Basden and Richard Myers. The durham adaptive optics real-time controller: capability and extremely large telescope suitability. *Monthly Notices of the Royal Astronomical Society*, 2012.

[7] L. Wang. Design and Testing of GPU based RTC for TMT NFIRAOS. In S. Esposito and L. Fini, editors, *Proceedings of the Third AO4ELT Conference*, page 17, December 2013.

[8] Antonin H. Bouchez, Richard G. Dekany, John R. Angione, Christoph Baranec, Matthew C. Britton, Khanh Bui, Rick S. Burruss, John L. Cromer, Stephen R. Guiwits, John R. Henning, Jeff Hickey, Daniel L. McKenna, Anna M. Moore, Jennifer E. Roberts, Thang Q. Trinh, Mitchell Troy, Tuan N. Truong, and Viswa Velur. The palm-3000 high-order adaptive optics system for palomar observatory. In *Proc. SPIE*, volume 7015, page 70150Z, 2008.

[9] Lianqi Wang and Brent Ellerbroek. Computer simulations and real-time control of elt ao systems using graphical processing units. In *Proc. SPIE*, volume 8447, page 844723, 2012.

[10] S. Mauch, J. Reger, C. Reinlein, M. Appelfelder, M. Goy, E. Beckert, and A. Tünnermann. Fpga-accelerated adaptive optics wavefront control. In *Proc. SPIE*, volume 8978, page 897802, 2014.

[11] Heng Zhang, Zoran Ljusic, Gary Hovey, Jean-Pierre Véran, Glen Herriot, and Maxime Dumas. A high-performance fpga platform for adaptive optics real-time control. In *Proc. SPIE*, volume 8447, page 84472E, 2012.

[12] Enrico Fedrigo, Robert Donaldson, Christian Soenke, Richard Myers, Stephen Goodsell, Deli Geng, Chris Saunter, and Nigel Dipper. Sparta: the eso standard platform for adaptive optics real time applications. In *Proc. SPIE*, volume 6272, page 627210, 2006.

[13] David Barr, Alastair Basden, Nigel Dipper, Noah Schwartz, Andy Vick, and Hermine Schnetler. Evaluation of the xeon phi processor as a technology for the acceleration of real-time control in high-order adaptive optics systems. In *Proc. SPIE*, volume 9148, page 91484B, 2014.

[14] Malcolm Smith, Dan Kerley, Glen Herriot, and Jean-Pierre Véran. Benchmarking hardware architecture candidates for the nfiraos real-time controller. In *Proc. SPIE*, volume 9148, page 91484K, 2014.

[15] Intel. Intel xeon phi coprocessor 5110p, November 2012.

[16] Jianbin Fang, Henk Sips, LiLun Zhang, Chuanfu Xu, Yonggang Che, and Ana Lucia Varbanescu. Test-driving intel xeon phi. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ICPE '14, pages 137–148, New York, NY, USA, 2014. ACM.

[17] John D. McCalpin. Stream: Sustainable memory bandwidth in high performance computers. Technical report, University of Virginia, 1991-2007. http://www.cs.virginia.edu/stream/.

[18] I. Z. Reguly, E. László, G. R. Mudalige, and M. B. Giles. Vectorizing unstructured mesh computations for many-core architectures. In *Proceedings of Programming Models and Applications on Multicores and Manycores*, PMAM'14, pages 39:39–39:50, New York, NY, USA, 2014. ACM.

[19] Kamran Karimi, Neil G Dickson, and Firas Hamze. A performance comparison of cuda and opencl. *arXiv preprint arXiv:1005.2581*, 2010.

[20] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[21] Alastair Basden, Deli Gang, Richard Myers, and Eddy Younger. Durham adaptive optics real-time controller. *Applied Optics*, 49(32), 2010.

[22] Intel. Intel xeon phi processor 7290f, 2016. Accessed: 2016-007-25.

[23] NVIDIA. Nvidia tesla p100, 2016. Accessed: 2016-07-25.

[24] NVIDIA. Nvidia tesla k80, 2016. Accessed: 2016-007-25.

[25] NVIDIA. Whitepaper nvidia tesla p100, 2017. Accessed: 2017-05-22.

[26] Malcom Smith. Thirty meter telescope narrow-field infrared adaptive optics system (nfiraos) real-time controller prototyping results. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 9909–202. International Society for Optics and Photonics, 2016.

Chapter 8

Conclusions and perspectives

8.1 Conclusion

In this thesis, we have tested two novel computational hardware that have the potential to be used for adaptive optics (AO) systems in the next generation of ground-based astronomy telescopes. We have focused our efforts on identifying hardware solutions for the most computationally intensive routines that are performed in the AO control loop, namely the wavefront processing unit (WPU) and the wavefront reconstruction. The hardware we selected were the Mellanox TILE-Gx36 (see chapter 6) and the Intel Xeon Phi, specifically the Knights Corner 5110p (see chapter 7). At a glance these technologies do not seem too different from each other, both being many-core co-processors. A detailed analysis of the characteristic shows obvious differences allowing them to fill different roles in the AO RTC.

The Mellanox TILE-Gx36 is a many-core co-processing card with four 10 GbE ports, designed for the quick processing of incoming data and designed for use within data centres. This makes it a great candidate for the WPU at the front-end of AO RTCs, which process large amounts of pixel data from the WFS(s). Its low memory bandwidth means it is not suitable for the wavefront reconstruction.

In chapter 6, we extensively tested the TILE-Gx36 for its suitability as a WPU in an AO RTC. We focused on the Shack-Hartmann (SH) wavefront sensor (WFS) although the results obtained can easily be applied to a pyramid WFS. We found that using the Zero Overhead Linux (ZOL) mode of operation, which removes the interaction of the OS with the running code, provides a real-time environment that has very little variation in the computation time. For example, when performing the calculations necessary in the WPU and using a 240×240 detector, the

jitter (i.e. standard deviation) can be as low as 1.27 μs giving a very deterministic calculation time.

From this detailed analysis we have shown that the TILE-Gx offers the necessary processing power as well as the very low jitter, required for a typical wavefront processing units of most first-light instrument on the E-ELT. We believe that the current version of the TILE-Gx and subsequent releases (with more cores and higher memory bandwidth) make this technology a very strong contender for most of the AO RTC systems planned for the ELTs.

In contrast the Intel Xeon Phi (Knights Corner) is much closer to a GPU with a high memory bandwidth and many-cores connected via a PCIe bus. The PCIe connection makes it harder to get large quantities of data into the card at high data transfer rates and in a reliable manner (meaning it is not suitable for a WPU). The high memory bandwidth and many-cores makes it a good candidate for the wavefront reconstruction stage.

The Xeon Phi from Intel has been tested in chapter 7 for its ability to offer the required computational power for the wavefront reconstruction, specifically the matrix-vector multiplication (MVM). The Xeon Phi is a co-processor similar to GPUs though it uses x86 instruction set cores, therefore simplifying code development. It offers a high memory bandwidth of 160 GB s^{-1} and 59 processing cores that can each run four threads allowing the accelerating of parallel code such as the MVM.

We demonstrated that while the Xeon Phi offers high performance and that the performance scales well with the number of Xeon Phis used, it has a high level of jitter. The mean execution time may be improved upon as we only tested a configuration where we wait for all

the slopes to be ready before starting. Better results could be obtained by pipelining the data and starting calculations as soon as a sufficient number of slopes are ready. We identified the main source of the jitter as being the data transfer in and out of the Xeon Phi. This level of jitter makes the current iteration of the Xeon Phi unusable in the hard real-time pipeline of an AO RTC. The next version of Xeon Phi (Knights Landing) offers increased memory bandwidths and more cores. It is no longer a co-processor but rather a CPU directly seated on the mother-board of a computer. This hopefully offers an increased computational power while at the same time reducing the jitter. In addition, this will make the implementation of a real-time OS on the Xeon Phi possible, further reducing the jitter. This Knights Landing has been released July 2016 and is undergoing testing by Durham University and NFI-RAOS teams. Initial results show very a promising behaviour, in-line with the predicted performance presented in this thesis.

Table 8.1 shows a summary of the results both the for TILE-Gx and for the Xeon Phi (including predictions for the Knights Landing). Results are presented for various detector sizes and various numbers of sub-apertures, the key parameters driving the complexity of calculations. We start with sizes that are consistent with current high-order AO systems in operation such as SPHERE and AOF (both using SPARTA as the RTC). To a first approximation, they can be compared to systems using a WFS detector with 200x200 pixels and 40×40 sub-apertures. We also show current ELT first-light instruments to highlight the fact that these technologies are capable of delivering the necessary performance.

Table 8.1: Summary of the main TILE-Gx and Xeon Phi performance results and predictions for the Knights Landing. All times are in μs. $\overline{t_{wp}}$ is the mean wavefront processing time. $\overline{t_{ro}}$ is the mean wavefront read-out time. $\overline{t_{wd}}$ is the mean wavefront delay (see section 5.1.2). nSubs is the number of sub-apertures on the WFS.

Wavefront processing (Tile-GX)				Wavefront reconstruction (Xeon Phi)		
Detector size	$\overline{t_{wp}}$	$\overline{t_{ro}}$	$\overline{t_{wd}}$	nSubs	Knights Corner	Knights Landing
200×200	44	40	4	40×40	210	27
400×400	172	166	6	60×60	689	127
640×640	475	428	48	80×80	1587	405
800×800	822	670	153	100×100	2755	997

Taken separately, these figures show the ability of the hardware to match specific problems. Table 8.2 illustrates the overall performance of a theoretical RTC system comprised of a single TILE-Gx and a single Xeon Phi. We have assumed that the TILE-Gx can be used in pipeline mode and that there is no data communication delay (e.g. additional time needed to transfer data between the WFU and the wavefront reconstruction hardware) or any other forms of temporal delays. Results are presented both for the Xeon Phi tested (i.e. Knights Corner) and the current released version at the time of writing (i.e. Knights Landing). These figures are not meant to give an absolute accurate representation of the system performance but to give a general indication of the performance these hardware would be able to achieve.

We see that the Knight Landing will be able to provide sub-millisecond latency for all ELT baseline systems (SCAO configuration) (see section 5.6). In AO configurations requiring more than one WFS or more than one DM, multiple sets of hardware could be used in parallel. The TILE-Gx would not interfere with other ones as the WPU units operate in-

dependently of one another. For the wavefront reconstruction step this can be distributed across multiple Xeon Phis with little penalty as has been shown in section 7.3.3.

To the best of our knowledge, we were the first groups to identify the potential TILE-Gx for AO RTC applications[1, 2]. The results presented show that this technology not only provides the performance required for WPUs but also provides it in a deterministic manner. We were also among the first groups to publish on the Intel Xeon Phi[3, 4, 1]. The Xeon Phi 5110p shows good overall performance but poor stability (i.e. jitter). The Knights Landing, the lastest Xeon Phi version at the time of writing, is showing very promising early results in improving both mean performance and jitter. For these reasons, we believe that both technologies deserve serious consideration for future AO RTC systems.

Table 8.2: Prediction of the mean overall execution time of AO RTC comprised a TILE-Gx for the wavefront processing and a Xeon Phi for the wavefront reconstruction. All times are in μs. Values are presented assuming the pipeline results from the TILE-Gx (see section 6.4).

Detector Size	nSubs	TILE-Gx + Knights Corner	TILE-Gx + Knights Landing
200×200	40×40	215	31
400×400	60×60	693	131
640×640	80×80	1591	409
800×800	100×100	2759	1001

8.2 Future work

The ELTs are fast approaching and shall be ready within the next decade. The development of the first light instruments is well underway.

The results presented in this thesis are presented as more of a guide or an indication of the performance of the hardware, not as the definitive performance of a full-fledged RTC. The effort of this thesis has been more focused on trying to isolate and identify the hardware with high potential. In a fully developed AO RTC there would be a more complex real-time pipeline with interactions between different processing stages, as well as interactions with the soft real-time cluster, that may not have been present in this investigation. In the work carried out here, we have tried to stay as generic as possible and therefore as far away as possible to a specific instrument or implementation. Only in taking into account instrument specific environments would we be able to provide an accurate estimate of the complete RTC chain.

While the results of the TILE-Gx presented in this thesis are extremely promising there is more work required to assess its ability as an AO RTC WPU. In the tests presented in chapter 6 an FPGA was used to emulate a camera. This allowed us to send known data in a deterministic manner to the TILE-Gx. These tests should be repeated with a real camera to determine if it could handle a real-camera load. In the tests we have not passed the resultant slope data out of TILE-Gx into another computer; this could be done either via the 10 GbE connection or the PCIe bus depending on requirements. This would need to be done to assess the true latency from the last pixel out of the camera read-out electronics to last slope out of the WPU.

The Xeon Phi tested in chapter 7 provided good overall performance as compared to its same generation competitors. Since then memory bandwidths of newer chips have increased with GPUs such as the P100 or Xeon Phis such as the Knights Landing. The major problem with the

Xeon Phi (Knights Corner) was the variations in latency. The Knights Landing on the other hand has both higher memory bandwidth and the potential of reducing variations in latency (i.e. jitter) since it is no longer a co-processor and is closer to a CPU.

With the next generation having been released with a much high memory bandwidth and as a CPU rather than a co-processor the Xeon Phi looks like a promising candidate. While we have predicted the potential performance of the Knights Landing results have not yet been published. Initial results presented at workshops from groups such as Durham University and the NFIRAOS consortium are very promising.

References

[1] David Barr, Alastair Basden, Nigel Dipper, and Noah Schwartz. Reducing adaptive optics latency using many-core processors. *Proc. AO4ELT4*, 1, 2015.

[2] David Barr, Noah Schwartz, Andy Vick, John Coughlan, Rob Halsall, Alastair Basden, and Nigel Dipper. Novel technology for reducing wavefront image processing latency. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 99094P–99094P. International Society for Optics and Photonics, 2016.

[3] David Barr, Alastair Basden, Nigel Dipper, Noah Schwartz, Andy Vick, and Hermine Schnetler. Evaluation of the xeon phi processor as a technology for the acceleration of real-time control in high-order adaptive optics systems. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 91484B–91484B. International Society for Optics and Photonics, 2014.

[4] David Barr, Alastair Basden, Nigel Dipper, and Noah Schwartz. Reducing adaptive optics latency using xeon phi many-core processors. *Monthly Notices of the Royal Astronomical Society*, 453(3):3222–3233, 2015.

Appendix A

Appendix

Contents

A.1 Publications

Monthly Notices
of the
ROYAL ASTRONOMICAL SOCIETY

MNRAS **453,** 3222–3233 (2015) doi:10.1093/mnras/stv1813

Reducing adaptive optics latency using Xeon Phi many-core processors

David Barr,[1,2★] Alastair Basden,[3] Nigel Dipper[3] and Noah Schwartz[1]

[1] *UK Astronomy Technology Centre, Royal Observatory, Blackford Hill, Midlothian, Edinburgh EH9 3HJ, UK*
[2] *Heriot-Watt University, Edinburgh Campus, Riccarton, Currie EH14 4AS, UK*
[3] *Durham University, Centre for Advanced Instrumentation, South Road, Durham DH1 3LE, UK*

Accepted 2015 August 4. Received 2015 August 3; in original form 2015 July 14

ABSTRACT

The next generation of Extremely Large Telescopes (ELTs) for astronomy will rely heavily on the performance of their adaptive optics (AO) systems. Real-time control is at the heart of the critical technologies that will enable telescopes to deliver the best possible science and will require a very significant extrapolation from current AO hardware existing for 4–10 m telescopes. Investigating novel real-time computing architectures and testing their eligibility against anticipated challenges is one of the main priorities of technology development for the ELTs. This paper investigates the suitability of the Intel Xeon Phi, which is a commercial off-the-shelf hardware accelerator. We focus on wavefront reconstruction performance, implementing a straightforward matrix–vector multiplication (MVM) algorithm. We present benchmarking results of the Xeon Phi on a real-time Linux platform, both as a standalone processor and integrated into an existing real-time controller (RTC). Performance of single and multiple Xeon Phis are investigated. We show that this technology has the potential of greatly reducing the mean latency and variations in execution time (jitter) of large AO systems. We present both a detailed performance analysis of the Xeon Phi for a typical E-ELT first-light instrument along with a more general approach that enables us to extend to any AO system size. We show that systematic and detailed performance analysis is an essential part of testing novel real-time control hardware to guarantee optimal science results.

Key words: instrumentation: adaptive optics.

1 INTRODUCTION

Adaptive optics (AO; Babcock 1953) is a technique used to mitigate atmospheric turbulence and partially restore the diffraction limit of optical and near-infrared ground-based astronomical telescopes, improving effective resolution to beyond the seeing limit. It is a critical technology for the next generation of Extremely Large Telescopes (ELTs) such as the European ELT (E-ELT) and essential to achieve high-angular resolution. The main science goals of the ELTs, such as high-redshift galaxies (Puech et al. 2010), stellar formation (Evans et al. 2011) or direct exoplanet imaging, provide challenging requirements that current AO systems are unable to meet. To achieve these requirements new hardware needs to be investigated and characterized in the context of ELT-scale AO systems.

AO systems use a corrective element, typically the surface of a deformable mirror (DM), to compensate for the phase retardation introduced by atmospheric turbulence. The required compensation is measured using one or more wavefront sensors, and must be applied within a short time-scale before the atmosphere has further evolved (i.e. a fraction of the coherence time) and is typically of the order of one millisecond. Because of the significant increase in the primary mirror size of ground-based telescopes, from the 4–10 m class telescopes of today; to the planned 30–40 m, the simple extrapolation of current real-time (RT) technology is not possible. Research and development of suitable AO technologies is required. The number of degrees of freedom of an AO system is proportional to the square of the primary mirror size and the next generation of ELTs will make computation, scaling with the fourth power of telescope diameter, extremely demanding. Real-time controllers (RTC) translating wavefront measurements into DM commands are at the heart of the AO system and therefore naturally one of the main areas of investigation. Table 1 shows the requirements for a selection of current and planned AO systems and stresses the high number of degrees of freedom (i.e. AO systems size) and the high update frequencies.

Wavefront reconstruction, translating measured wavefront into new DM commands, is by far the most computationally intensive algorithm that an ELT-scale RTC is required to perform. The most common wavefront reconstruction algorithm used is the matrix–vector multiplication (MVM). The DM commands d are related to

★ E-mail: David.Barr@stfc.ac.uk

Table 1. Selection of current and proposed AO systems with demanding computational requirements for RT control systems. SPHERE (Sauvage et al. 2010), GRAAL (Paufique et al. 2012), PALM3000 (Bouchez et al. 2009), GPI (Pazder et al. 2012), NFIRAOS (Boyer et al. 2014), HARMONI (Fusco et al. 2010), EAGLE (Basden et al. 2012), EPICS (Vérinaud et al. 2010).

System	AO type	Frequency	WFS	# DM actuators
SPHERE (VLT)	XAO	1.2 kHz	40 × 40	41 × 41
GRAAL (VLT)	GLAO	1 kHz	4 × (40 × 40)	1170
PALM3000 (Hale)	XAO	2 kHz	63 × 63	(16 × 16) + (66 × 66)
GPI (Gemini)	XAO	1.2 kHz	64 × 64	64 × 64
NFIRAOS (TMT)	MCAO	0.8 kHz	7 × (60 × 60)	(63 × 63) + (76 × 76)
EAGLE (E-ELT)	MOAO	0.25 kHz	11 × (84 × 84)	20 × (85 × 85)
HARMONI (E-ELT)	SCAO	0.5 kHz	84 × 84	85 × 85
EPICS (E-ELT)	XAO	2–3 kHz	210 × 210	211 × 211

the slopes s through a linear equation $d = \mathbf{G}^{-1}s$, where $\mathbf{G}^{-1}$ is the control matrix. The computational complexity for an MVM grows as $\mathcal{O}(M^2)$, where M is the number of degrees of freedom of the AO system.

In recent years, several novel computationally efficient wavefront reconstruction algorithms have been developed (Poyneer, Gavel & Brase 2002; Rosensteiner 2012; Véran et al. 2014). These alternative wavefront reconstruction approaches are typically iterative and generally unable to efficiently take advantage of modern multicore and many-core hardware architectures. Although the MVM typically has the largest requirements in terms of number of operations and memory usage compared to these other methods, it is highly parallelizable making efficient use of modern multicore architectures and widely used by the AO community. For small AO systems, CPU-based systems can typically be used. As we move towards ELT-scale systems, calculations become more and more difficult to handle with CPU alone, limited both by available memory bandwidth, and raw processing power.

To achieve the required computational power, many groups have focused on GPUs (Bouchez et al. 2008; Basden & Myers 2012; Gratadour et al. 2012; Wang & Ellerbroek 2012; Wang 2013; Sevin et al. 2014) since they offer a potential suitable parallel environment to reduce the latency associated with the MVM calculation. Field programmable gate arrays (FPGAs) have also been used, although typically for smaller systems (Mauch et al. 2014), or for only a section on the AO control loop (Zhang et al. 2012) or limited to the pixel processing as part of heterogeneous RTC hardware (Fedrigo et al. 2006). These hardware accelerators generally suffer from the same disadvantages: limited data transfer into and out of the accelerator. They lead to complex heterogeneous computing environments which give rise to complex memory structures and the movement of large quantities of data between different computational components. Accelerator architectures traditionally evolve quickly as new hardware is released, which may not be compatible with older systems, leading to lifetime and portability issues. This can cause long development times and difficulty in maintaining and upgrading systems.

In this paper, we investigate the performance of the Intel Xeon Phi for wavefront reconstruction using the MVM algorithm. The Xeon Phi uses x86 instruction set microprocessors (same as conventional CPUs), which may help in lowering the barriers to entry compared with GPUs or FPGAs, i.e. no specialized code base or application programming interface (API) is required. The implemented code can easily be modified and upgraded, should a more performant hardware be released. The Xeon Phi also offers high-memory bandwidth to accelerate memory-bound parallel algorithms. It is however, designed for high-performance computing where the requirements are more focused on the mean execution time rather than on the determinism of execution time. A detailed analysis of performance in a realistic AO environment is therefore essential. Previous investigations were limited and focused on non-real-time (non-RT) Linux systems (Barr et al. 2014) or on a very specific AO system (Smith et al. 2014) making the generalization to other systems difficult. In addition, a detailed analysis of the timings is crucial to fully understand the limitations of the hardware and extrapolate to future hardware developments. Different science cases will have different tolerances on the acceptable jitter (variation in execution time) or outliers (results significantly apart from the mean) for example, which may or may not impact science results significantly.

The rest of the paper is organized as follows. In Section 2, we give an overview of the RTC architecture including the different timings and performance metrics used for the investigation. In Section 3, we discuss the actual performance of the Xeon Phi and present detailed results of timings for a standalone system as well as integrated into an RTC system. Results are presented for a typical first-light E-ELT instrument and also for a more general approach, in order to extrapolate to any AO system size. In Section 4, we look at the expected Xeon Phi developments and anticipate potential performance. Finally, in Section 5 we draw our conclusions.

2 XEON PHI BASED RT CONTROL

2.1 RTC architecture

The Xeon Phi is a many-core accelerator co-processor card connected to a processor via a PCIe bus offering a high level of programmability (standard C/C++ with compiler assisted offload), high throughput, high performance per watt and low cost. The main disadvantage, as with most accelerators, is that data communication between the host computer and the Xeon Phi will add unwanted delays (and jitter) to the AO loop. This leads to a heterogeneous computing environment which may cause issues with complex memory management and makes optimization difficult. The Xeon Phi is similar in that sense to general purpose GPUs used in high-performance computing environments. The Xeon Phi differs however from GPUs by offering x86 instruction set cores, allowing programmers to use the same design techniques as they would with CPUs. This has the potential to speed up development time and does not require specialist knowledge of programming paradigms and toolkits such as CUDA or OPENCL.

The Xeon Phi model under investigation here is the 5110P, which offers 60 cores, a clock speed of 1 GHz, 8 GB of GDDR5 memory and has a maximum theoretical memory bandwidth of 320 GB s^{-1}. For detailed specifications of the Xeon Phi and host computer used in this paper see Table A1 in Appendix A. The Xeon Phi 5110P clock speed is slower than that of modern CPUs which can typically reach 3–4 GHz. This suggests that the Xeon Phi would be unable

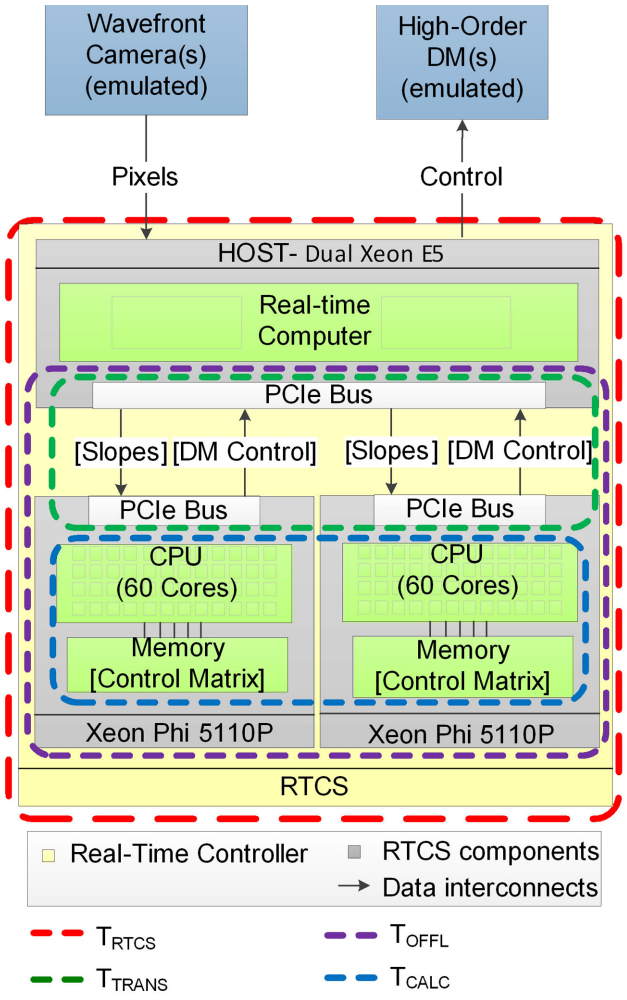


Figure 1. Hardware configuration used to benchmark the Xeon Phi. Two Xeon Phis are connected to the host computer (Xeon E5-2650) via PCIe bus. The Xeon Phi is used to accelerate the MVM. The control matrix is stored before calculations in the Xeon Phi's memory, while wavefront slopes and DM commands vectors are transferred at each AO frame. The dashed boxes represent details of the four different times that are investigated.

to compete for performance on sequential code. Given the number of cores and the high-memory bandwidth, it has the potential to outperform current CPUs on parallel codes such as the MVM.

The aim of this investigation is to characterize and benchmark the Xeon Phi in the context of a low latency and low jitter AO control loop. We focus our study on the main task of the RTC which is to control the core AO system. We do not include additional tasks (often characterized as soft RT tasks) and do not relate our findings to a specific instrument design, preferring to adopt a more general approach. The Xeon Phi is used to accelerate MVM calculations and the rest of the AO processing tasks (such as image calibration, slope calculation and DM control laws) are performed by the host computer. The hardware configuration is shown in Fig. 1. The host computer, composed of a Dual Xeon E5-2650, receives wavefront camera(s) pixels (typically a Shack–Hartmann sensor) and calculates the slopes. In the studied configuration, the wavefront camera data is emulated and not physically connected to any hardware, ensuring that the system is not limited by the camera's frame rate. The Xeon Phi then receives the slopes from the host computer through

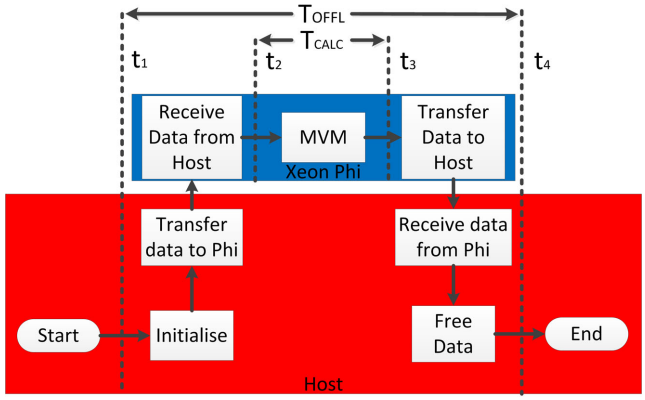


Figure 2. Diagram showing a simplified version of the implemented process and the four different timings measured to benchmark the Xeon Phi. t_1 and t_4 are taken on the Xeon E5; t_2 and t_3 are taken on the Xeon Phi. Due to the Xeon Phi and the Xeon E5 having separate unsynchronized clocks, only the total data transfer time is accessible and is calculated from $T_{\mathrm{OFFL}} - T_{\mathrm{CALC}}$.

a PCIe bus and computes the command vector that is finally sent to the DM(s). Equally, no DM is physically connected to the host PC.

The Xeon Phi and the Xeon E5 have a separate clock, making it difficult to accurately time data transfer and MVM calculations using the same clock. In order to produce accurate timings, we measure the difference between times on the CPU and times on the Xeon Phi, removing issues of asynchronous clocks. Throughout this paper, we will investigate four different timings to fully benchmark the performance of the Xeon Phi.

(i) T_{OFFL} is the offload time: this represents the time from the first data sent from the host computer to the Xeon Phi(s) to the last data received back on the host computer. This includes data transfer (i.e. slope and DM command vectors) and MVM calculation on the Xeon Phi(s). $T_{\mathrm{OFFL}} = t_4 - t_1$ (see Fig. 2).

(ii) T_{CALC} is the calculation time: this refers to the time taken for the MVM to be calculated on the Xeon Phi (or Xeon Phis) excluding any transfer times. $T_{\mathrm{CALC}} = t_3 - t_2$ (see Fig. 2).

(iii) T_{TRANS} is the combined transfer time: this represents the time taken for the data to be transferred in and out of the Xeon Phi. This encapsulated both the transfer of the slope vector and DM commands. $T_{\mathrm{TRANS}} = T_{\mathrm{OFFL}} - T_{\mathrm{CALC}}$ (see Fig. 2).

(iv) T_{RTCS} is the RT control time: this represents the time taken for an entire AO frame to execute. This includes the wavefront sensor (WFS) pixel processing, slope computation, the MVM calculation on the Xeon Phi(s) as well as any additional background tasks of the RT control system.

Data are taken according to the scheme presented in Fig. 2. At the start of the process, we initialize and pre-load the control matrix $\mathbf{G}^{-1}$ into the Xeon Phi memory. The slope vector is then transferred to the Xeon Phi (marked as time t_1) and the MVM calculation starts (time t_2). At the end of the calculation (time t_3), we transfer the result back to the host computer (time t_4) and loop back to the slope transfer. After a statistically significant number of timings (typically 10^6), we free the memory and end the process. Timings t_1 and t_4 are taken on the host computer whereas t_2 and t_3 are taken on the Xeon Phi. This timing scheme allows us to time the overall offload time T_{OFFL} and MVM calculation time T_{CALC} separately. From measuring T_{OFFL} and T_{CALC}, the combined transfer time T_{TRANS} can be derived. Separating between transfer and calculation times offers

us the capability to locate where additional time delays are being generated.

2.2 RT Linux and Xeon Phi optimization

Modern computers are generally not RT processors and operating systems have background processes running which affect determinism. AO systems need a high level of determinism which non-RT operating systems are generally unable to provide. RT Linux on the other hand, gives us greater control on the order (priority) in which processes are carried out by the operating system. These processes with raised priority are able to pre-empt the lower priority tasks to give greater control and predictability in execution time. For the host computer, both a non-RT and an RT Linux kernel will be investigated. An RT pre-empt 3.10 Kernel was installed on the host computer which is running RED HAT 6.4. The RT Linux kernel was not edited nor modified. The Xeon Phi itself is running a non-RT micro-operating system based on a Linux kernel.

To perform the MVM calculation, we investigated the performance of the Intel MATH KERNEL LIBRARY (MKL), the MAGMA library (Bosma, Cannon & Playoust 1997) and an MVM code developed in-house which uses OPENMP for parallelization. MAGMA offered much more abstraction from the Xeon Phi architecture than the other two, which allows quick development but at the cost of reduced performance. The in-house code was optimized for certain matrix sizes and was able to outperform MKL on some specific cases. In general, MKL gave a high baseline performance for all cases. It was decided to use MKL and focus on general performance of the Xeon Phi rather than focus on specific cases where in-house code can be tuned and optimized to obtain the best performance. Using MKL has the double advantage of reaching very good performance overall but also ensuring that simple software design techniques can be used without requiring in-depth knowledge of the Xeon Phi architecture.

From a previous study (Barr et al. 2014), we have shown that the Xeon Phi performs best when the problem size can fit its architecture. The Xeon Phi 5110p has 60 cores, with one core reserved for input/output routines. Each core can support up to four threads, which means that when the problem size is divisible by 236 (i.e. 4×59), optimal performance is obtained. This difference in performance is likely due to the architecture and also how the MKL library handles the parallel sections of code. When the problem size fits the architecture, the library is able to split the problem evenly between all cores. When this is not the case, the library has to perform some dynamic resource handling that brings in more overhead, degrading performance. When the problem size is not divisible by 236, the control matrix is therefore padded with zeros to fit the architecture and reach the best achievable performance.

3 BENCHMARKING THE XEON PHI

In this section, we present a detailed analysis of the performance of the Xeon Phi. It is important here to note that different science cases will put different constraints on AO system requirements. Some (e.g. direct exoplanet imaging) will require very high image contrasts. Achieving these contrast levels will require very low and stable AO loop latencies. Other science drivers (e.g. high-redshift galaxies, stellar formation) will have somewhat lower requirements, in particular on stability. The variation in latency (i.e. stability of the system), often called jitter, will be evaluated in this paper as the standard deviation[1] of the measured times t. Outliers (i.e. frames taking significantly longer to complete than the mean execution time) are also crucially important. In order to refine the analysis, the full distribution of the measured timings will be given, along with mean, jitter, minimum, maximum and percentiles of timings completed before a given time.

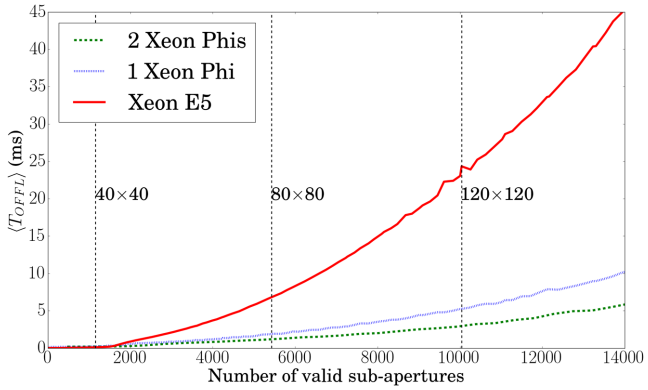


Figure 3. Comparison of the mean offload time $\langle T_{\rm OFFL} \rangle$ as a function of the number M of valid sub-apertures using 10^4 samples. Red: Xeon E5; Blue: single Xeon Phi; Green: two Xeon Phis. Results obtained using an RT Linux kernel. The dashed vertical lines represent the approximate size of an AO system in total number of sub-apertures.

In order to stay as general as possible, and not tie this study to any specific instrument design, the AO system size (M) will be defined as the total number of wavefront measurement points (typically the number of valid sub-apertures of a Shack–Hartmann wavefront sensor). The slope vector size is 2M as it contains the slopes in X and Y directions for each valid sub-aperture. In a typical single conjugate AO (SCAO) system, the number of valid sub-apertures is approximately equal to the number of DM actuators and so the DM command vector will be approximately half the size of the slope vector. The control matrix ($\mathbf{G}^{-1}$) will therefore be of dimension $\mathrm{M} \times 2\mathrm{M}$ unless explicitly stated otherwise.

After testing the Xeon Phi as a function of AO system size M, we focus our attention on a typical first-light SCAO E-ELT instrument with approximately 80×80 sub-apertures. For such a system, the control matrix contains 9440 rows and 5428 columns (taking into account an aperture with central obscuration), which in turns corresponds to a memory size of ≈205 MB, using 4 byte (32 bit) single-precision floating-point numbers. Typically slope data are only accurate to 16 bits, allowing 4-byte floats to provide sufficient accuracy for non-high-contrast AO systems (Basden, Myers & Butterley 2010a). Tests are run both on a non-RT Linux kernel and an RT Linux kernel system using one and two Xeon Phis, allowing us to investigate scaling with number of co-processors. Finally, we will show that the generality of our conclusions will not be lost by exploring a specific system size.

3.1 Influence of AO system size

3.1.1 Offload time as a function of AO system size

Fig. 3 shows the comparison of mean offload time $\langle T_{\rm OFFL} \rangle$ (i.e. including MVM calculation and data transfer) for the Xeon E5

[1] $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(t_i - \mu)^2}$, where the mean $\mu = \frac{1}{N}\sum_{i=1}^{N} t_i$ The measured distributions are not normal distributions and classical interpretation of the standard deviation is not possible.

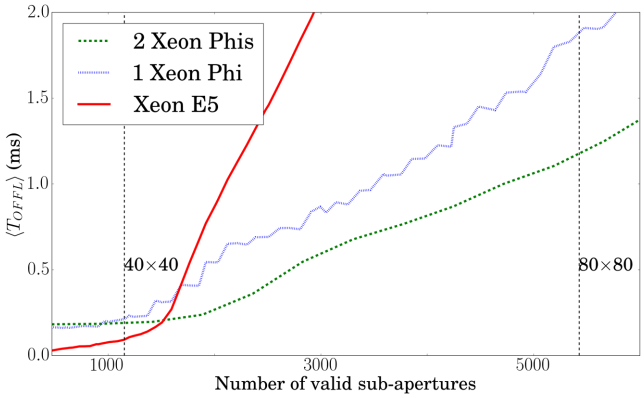


Figure 4. Comparison of the mean offload time $\langle T_{\mathrm{OFFL}} \rangle$ as a function of the number M of valid sub-apertures using 10^4 samples. This figure is a zoom of data found in Fig. 3 for better readability.

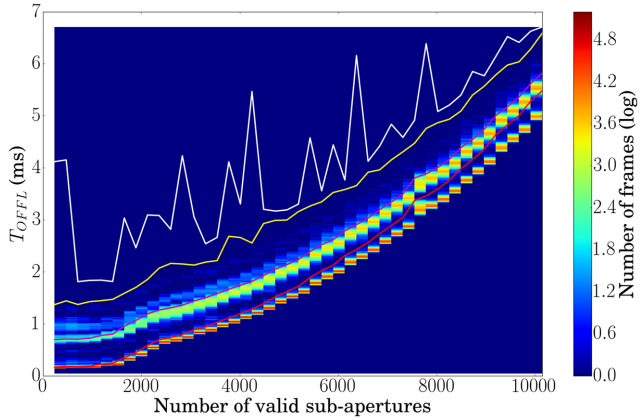


Figure 5. Variation in offload time T_{OFFL} using a single Xeon Phi with the host running an RT Linux Kernel. Only multiples of 236 are shown as they provide best performance. A logarithmic colour lookup table is use to visualize both peaks and tails of the distributions. The red line represents the mean, purple represents $P_{99\,\mathrm{per\,cent}}$, yellow represents $P_{99.99\,\mathrm{per\,cent}}$ and the white line represents the maximum time measured.

alone, a single Xeon Phi and two Xeon Phis used as accelerators as a function of AO size. For the smaller AO systems where the number of valid sub-apertures is less than approximately $M < 1500$ (such as for a 40 × 40 SCAO system) the Xeon E5 clearly outperforms the Xeon Phi(s). This is due to the need to transfer data (i.e. slope and DM command vectors) over the PCIe bus. Once the number of valid sub-apertures becomes larger, the Xeon Phis provide lower mean latencies (this clearly visible in Fig. 4, a zoomed version of Fig. 3 for the range 0–6000 sub-apertures).

As expected, for large numbers of valid sub-apertures $\langle T_{\mathrm{OFFL}} \rangle$ grows as $\mathcal{O}(M^2)$ for all devices, dominated by computation time. When two Xeon Phis are used, $\langle T_{\mathrm{OFFL}} \rangle$ can be further reduced; the difference is more clearly visible for large AO systems.

As stated previously, the mean execution time is insufficient to fully characterize the AO RTC. Fig. 5 shows the distribution of offload times T_{OFFL} as a function of AO system size, using an RT Linux kernel, and measuring T_{OFFL} over 10^6 iterations for each system size. Only matrix size multiples of 236 are shown because mean and variation in execution time are both increased for non-multiples. It can be seen that not only all system sizes have a similar bimodal distribution but the mean, the position of peaks and

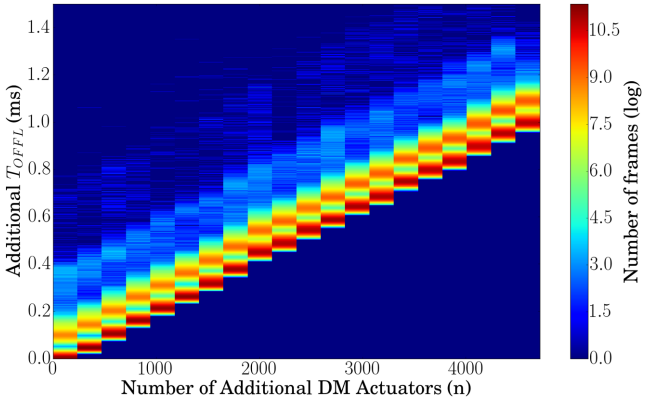


Figure 6. Increase of offload time T_{OFFL} as the number of elements of the control matrix is increased from $M \times 2M$ to a square with $2M \times 2M$ elements (the control matrix has $(M + n) \times (2M)$ elements, where $0 \leq n \leq M$ and $2M = 9440$). Data calculated using 10^6 samples. A logarithmic colour lookup table is use to visualize both peaks and tails of the distributions.

the different percentiles calculated (percentile of offload times that are completed by the given time) all follow a similar trend. Only the maximum offload time is shown to be somewhat irregular; we believe this is because of the limited number of samples used (i.e. 10^6) to calculate the distribution. From Fig. 5, we can legitimately say that studying a specific AO system size in detail will not remove the generality of the analysis as results can be scaled to match the desired system size. Results obtained with two Xeon Phis (data not shown here) show the same scalability.

3.1.2 *Influence of shape of the control matrix*

So far we have discussed the performance of the Xeon Phi in the context of SCAO systems, where the number of DM actuators is approximately equal to the number of sub-apertures and where we have assumed that the control matrix shape is $M \times 2M$. In this section, we investigate how the shape of the control matrix affects the general performance of the MVM calculation. Fig. 6 shows the additional offload time T_{OFFL} taken as we increase the control matrix size from $M \times 2M$ to a square with $2M \times 2M$ elements (i.e. the control matrix has $(M + n) \times (2M)$ elements, where $0 \leq n \leq M$). The baseline system (i.e. where $n = 0$) is equivalent to an 80 × 80 AO system with $2M = 9440$. As we have stated previously, all dimensions of the matrix are a multiple of 236 to maximize performance (explaining the steps in performance for every increase of n by 236). We see that the time increase is linear (i.e. increases linearly with the number of additional control matrix elements) and the overall shape of the distribution remains identical for all system sizes (double peak with a tail). It is important to stress at this point that using Fig. 5 in conjunction with Fig. 6 enables us to estimate the performance of a single Xeon Phi for most AO system sizes, regardless of the size or shape of its control matrix.

3.1.3 *Memory bandwidth*

When performing the wavefront reconstruction in an AO RTC using an MVM algorithm, the input vector is updated at every iteration (typically from hundreds to thousands of times per second), while the control matrix will remain constant for periods of time between tens of seconds to several hours. However, for large AO systems the matrix is too large to be stored in cache, and so must be read from

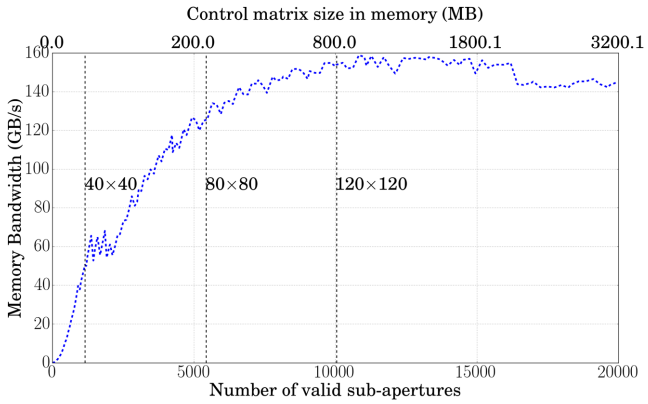


Figure 7. Memory bandwidth for a single Xeon Phi performing an MVM as a function of number of valid sub-apertures. This includes the data transfer time across the PCIe bus. The calculation only memory bandwidth will be slightly higher.

memory at each iteration. Therefore, memory bandwidth becomes a performance limiting factor. CPU-based systems typically have large banks of DDR3 memory which are relatively slow. The Xeon Phi has access to faster GDDR5 and has a maximum theoretical memory bandwidth of 320 GB s^{-1}. In practice, the Xeon Phi appears to have a read memory bandwidth of 164 GB s^{-1} and a write memory bandwidth of 76 GB s^{-1} (Fang et al. 2014).

Here, we have measured computation time for the MVM operation, and use this information to calculate the achieved memory bandwidth. Fig. 7 shows memory bandwidth as a function of AO size (and the control matrix size stored in memory) for a single Xeon Phi calculated from the offload time $\langle T_{\rm OFFL}\rangle$. $\langle T_{\rm OFFL}\rangle$ includes the $\langle T_{\rm CALC}\rangle$ and $\langle T_{\rm TRANS}\rangle$ so the memory bandwidth of just the Xeon Phi processor will be slightly higher. For large AO systems, the calculation time is limited by the memory bandwidth, which peaks at about 160 GB s^{-1}, in agreement with Fang et al. (2014). We note that for smaller systems, memory bandwidth is not the performance limiting factor; however, these systems are not of interest here since we are concentrating on larger systems.

When the control matrix is larger than the Xeon Phi L2 cache (30 MB), we see a drop in memory bandwidth due to the processor having to transfer all or part of the control matrix from the slower GDDR5 memory. As we increase the size of the control matrix, the processors have to make more and more calls to the slower GDDR5 memory. This continues until the control matrix reaches around 800 MB where the memory bandwidth levels around 160 GB s^{-1}. At this point, the control matrix is significantly larger than the L2 cache and most memory access is with the GDDR5 memory. The MVM like other BLAS-1[2] or BLAS-2[3] routines is memory bandwidth limited.

To confirm the achievable memory bandwidth, we used the industry standard STREAM memory benchmarking (McCalpin 1991-2007) on both the Xeon E5 and the Xeon Phi. We compared the memory bandwidth to a TRIAD[4] test which is the STREAM benchmarking scheme most closely resembling an MVM operation. The Xeon E5 achieved a peak memory bandwidth of 63.7 GB s^{-1} and the Xeon Phi of 166.5 GB s^{-1}. Other groups have published similar results (Fang et al. 2014).

[2] Vector–vector operations.
[3] Matrix–vector operations.
[4] The Triad test involves the addition of two vectors (b and c) one vector multiplied by a scaling factor (q) $a_i = b_i + q \times c_i$.

Table 2. A comparison of advertized and achieved memory bandwidths for Dual Xeon E5-2650, NVidia K40 GPU and Xeon Phi 5110p.

	Dual Xeon E5-2650	NVidia K40	Xeon Phi
Advertized Max. GB s^{-1}	2 × 51.20	288	320
STREAM GB s^{-1}	63.7	229	166.5
Percentage	62.2 per cent	79.5 per cent	52.03

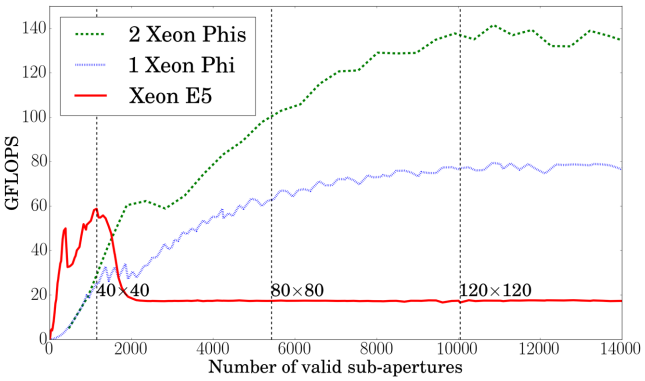


Figure 8. Number of FLOPS achieved during MVM for a dual Xeon E5-2650, one Xeon Phi and two Xeon Phis.

Table 2 compares the advertized (theoretical), achieved memory bandwidths using STREAM as well as the percentage of the advertized that was attained. The results are shown for the Dual Xeon E5-2650, the Xeon Phi 5110p as well as the NVidia K40 GPU (Reguly et al. 2014, a GPU released at around the same time as the Xeon Phi, which enables a direct comparison between hardware of the same generation). It can be seen that although the Xeon Phi has a higher theoretical maximum, the GPU can achieve a higher percentage of the advertized bandwidth than either the Xeon E5 or the Xeon Phi.

3.1.4 Floating-point operations per seconds

Floating-point operations per seconds (FLOPS) is a common metric that is frequently used to assess and compare performance of computing hardware. It can be calculated theoretically from equation (1),

$$\frac{\text{FLOPS}}{\text{cycle}} \times \frac{\text{cores}}{\text{socket}} \times \#\text{sockets} \times \text{clock}. \quad (1)$$

The Xeon Phi is advertized to be able to reach 1.011 TFLOPS. This value is far larger than we could expect to reach with the MVM or any other BLAS-1 or BLAS-2 operations due to the memory bandwidth limitations.

Fig. 8 shows the number of FLOPS that the Xeon E5-2650, Xeon Phi and two Xeon Phis have achieved when performing the MVM algorithm as function of AO system size. A peak in performance is seen for the Xeon E5-2650 below 2000 sub-apertures, which is when the matrix no longer fits in cache memory of the CPU. Curves for the single and dual Xeon Phi follow that of the memory bandwidth.

Table 3 shows the number of FLOPS that the Xeon E5-2650, Xeon Phi and two Xeon Phis have achieved when performing the MVM algorithm for a selection of AO system sizes. It can be seen that the performance of the Xeon Phi is much lower than the advertized FLOPS. This is what is expected due to the memory bandwidth attainable on the Xeon Phi. We see that using a second Xeon Phi

Table 3. Comparison of FLOPS for Dual Xeon E5-2650, single Xeon Phi and a Dual Xeon Phi system. Max is the maximum GFLOPS achieved across entire tested range, 0–14 000 sub-apertures.

GFLOPS of	40 × 40	80 × 80	120 × 120	Max
Xeon E5-2650	58.7	17.3	17.3	58.7
One Xeon Phi	25.2	62.7	76.9	79.4
Two Xeon Phis	29.0	100.3	136.9	141.5

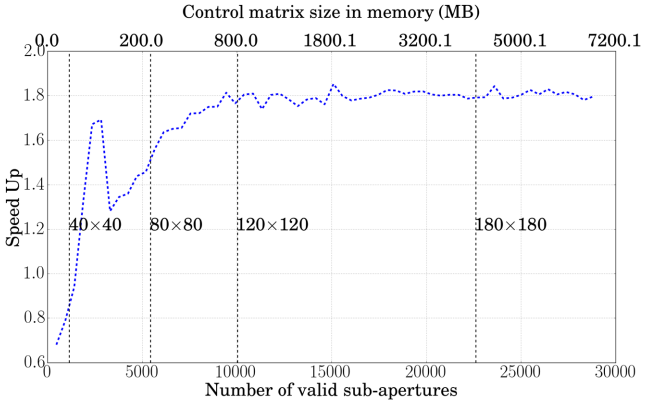


Figure 9. Relative performance of using two Xeon Phis instead of one: $\langle T_{\mathrm{OFFL}}\rangle^{2\mathrm{Phis}}/\langle T_{\mathrm{OFFL}}\rangle^{1\mathrm{Phi}}$ as a function of system size. The dashed vertical lines represent the approximate size of an AO system in total number of sub-apertures.

allows for a doubling in the number of FLOPS for large system, which is what would be expected.

3.1.5 Multiple Xeon Phi speedup

An MVM operation is highly parallelizable, and therefore easy to split between multiple Xeon Phi accelerators. Using a second Xeon Phi to complete the MVM doubles the available compute power and memory bandwidth. However, synchronization between the two processes makes achieving a speedup of ×2 difficult. Fig. 9 shows the offload time speedup that can be achieved by using two Xeon Phis instead of one. For small systems, using two Xeon Phis is actually slower than just relying on one, due to overheads. As the control matrix size increases, the speedup gradually rises to reach a plateau (starting from about a 120 × 120 AO system) of approximately 1.8. Using a second Xeon Phi to complete the MVM allows us to have twice the cache memory (60 MB). This explains the first performance peak on Fig. 9 while a single Xeon Phi would have to access the slower GDDR5 memory, the two Xeon Phis are able to fit the control matrix within the available combined cache memory. A single computer can contain up to eight Xeon Phis, as long as it has a sufficient number of PCIe bus lanes. Data can be transferred simultaneously to multiple Xeon Phis as long as there are unused lanes available.

3.2 Detailed analysis of temporal behaviour

As we have seen in the previous section, mean offload time does not enable us to fully characterize where the different latencies in using the Xeon Phi are coming from and how they affect the calculation speed. Access to the full distribution of computation times is therefore necessary. In addition, we have checked that studying a specific AO system size will not remove the generality

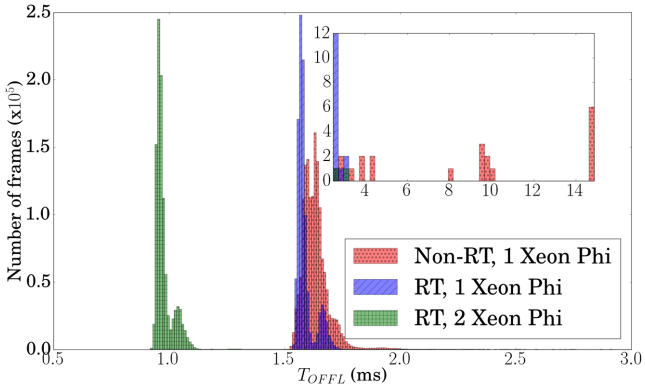


Figure 10. Histogram comparing the offload time T_{OFFL} for a 80 × 80 sub-aperture system calculated using 10^6 samples. T_{OFFL} encapsulates both calculation time and transfer time. Blue: single Xeon Phi on a RT Linux host; Red: single Xeon Phi on non-RT Linux; Green: two Xeon Phi on RT Linux kernel. Inset shows data from 2.5 to 15 ms with the number of frames ranging from 0–12 (showing outliers more clearly).

Table 4. Offload times T_{OFFL} corresponding to Fig. 10. $P_{N\,\mathrm{per\,cent}}$ is the Nth percentile of offload times that are completed by the given time. All times given in milliseconds.

T_{OFFL} (ms)	One Xeon Phi non-RT	One Xeon Phi RT	Two Xeon Phis RT	Xeon E5 RT
Jitter (σ)	0.066	0.039	0.0426	0.057
Min	1.514	1.525	0.918	3.480
Mean ($P_{XX\,\mathrm{per\,cent}}$)	1.631 ($P_{56\,\mathrm{per\,cent}}$)	1.587 ($P_{73\,\mathrm{per\,cent}}$)	0.978 ($P_{71\,\mathrm{per\,cent}}$)	3.622 ($P_{60\,\mathrm{per\,cent}}$)
$P_{99\,\mathrm{per\,cent}}$	1.863	1.704	1.097	3.661
$P_{99.9\,\mathrm{per\,cent}}$	2.009	1.765	1.320	3.865
$P_{99.99\,\mathrm{per\,cent}}$	2.099	2.198	1.678	4.035
$P_{99.999\,\mathrm{per\,cent}}$	4.295	2.663	2.118	15.394
Max	14.861	3.085	3.119	32.170

of the analysis as results can be scaled to match the desired system size. In this section, we analyse detailed results for a typical E-ELT first-light AO instrument with 80 × 80 sub-apertures (using a 9440 × 5428 element control matrix, or 205 MB).

3.2.1 Variability in offload time: T_{OFFL}

Fig. 10 shows T_{OFFL} for three tested configurations (i.e. one Xeon Phi used with a non-RT Linux host, and one and two Xeon Phi used with an RT Linux host). For each configuration, 10^6 measurements were taken. Table 4 shows more details on the offload time, analysing the distributions in terms of minimum and maximum values, jitter, mean and percentiles. The percentiles $P_{XX\,\mathrm{per\,cent}}$ are defined as the time by which XX per cent of the samples are completed. In other words $P_{99.99\,\mathrm{per\,cent}}$ represents a 1 in 10 000 event. Also shown in Table 4 are the results of the Xeon E5 completing the same MVM calculation on an RT Linux system.

Fig. 10 distribution is double peaked with a long tail due to outliers; Smith et al. (2014) have published a similar distribution. The shape of this distribution causes the mean not to sit at $P_{50\,\mathrm{per\,cent}}$ (the median value) but at $P_{56\,\mathrm{per\,cent}}$ for non-RT Linux and at $P_{73\,\mathrm{per\,cent}}$ for RT. Using an RT Linux does not appear to greatly decrease the mean latency when compared to the non-RT Linux. It seems however to reduce the majority of the extreme outliers and significantly

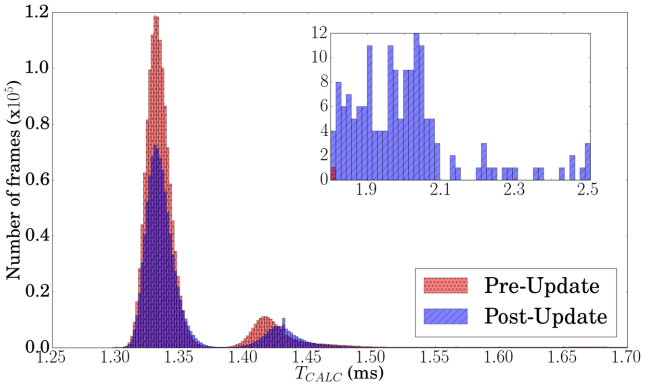


Figure 11. Histogram comparing the calculation time $T_{\rm CALC}$ for a 80 × 80 sub-aperture system calculated using 10^6 samples. $T_{\rm CALC}$ excludes the transfer time. Blue: single Xeon Phi post-firmware update; Red: single Xeon Phi pre-firmware update. Inset shows data from 1.8 to 2.5 ms with the number of frames ranging from 0–12 (showing outliers more clearly).

lower the jitter (which is defined as the deviation σ) from 0.066 to 0.039 ms. As expected, two Xeon Phis on an RT Linux system produces the lowest latency; $\langle T_{\rm OFFL}\rangle$ is about 1.6 times less than on a single Xeon Phi. However, jitter is increased. This is not entirely surprising as to complete the MVM, both Xeon Phis need to have finished their calculation. This means that for a given frame, jitter is introduced by the worst-performing Xeon Phi. The outliers occur so infrequently that $\langle T_{\rm OFFL}\rangle$ is unaffected.

Even by using two Xeon Phis, the number of outliers measured may still be a concern for certain AO configurations. With 1 outlier every 10 000 frames (i.e. $P_{99.99\,\rm per\,cent}$) for a system running at 500 Hz this will occur 180 times over the course of an hour observation. To identify where these outliers are arising and see if they can be reduced, we investigate in the following sections the split of $T_{\rm OFFL}$ into its component $T_{\rm CALC}$ and $T_{\rm TRANS}$.

3.2.2 *Variability in calculation time:* T_{CALC}

Since the MVM is only calculated on the Xeon Phi, it does not directly interact with the operating system running on the host computer. We do not expect to see any changes in the distribution of the MVM calculation time $T_{\rm CALC}$ on RT or non-RT systems.[5] However compiling the Xeon Phi drivers to be compatible with an RT Linux meant upgrading the Manycore Platform Software Stack (MPSS) to the latest version at the time of writing.[6] This update brought updated versions of Flash and System Management Controller (SMC). MPSS and the updates are only partially open source. This suggests that issues arising from updates may be solvable by editing these sections of the source code without the manufacturer's support. However, some sections are closed source which may make user modifications more difficult.

Fig. 11 shows the results for pre-updated Flash/SMC and after the update was applied. It seems to suggest that the update caused a reduction in performance and larger variations in timings. The variation in calculation time $T_{\rm CALC}$ is probably due to the fact that the Xeon Phi is running a non-RT micro-Linux which results in some large outliers (Véran et al. 2014). In Table 5, we see that the update slightly reduces both mean calculation times and jitter. It is

[5] This was shown to be true by going back to a non-RT system after update.
[6] MPSS 3.4 (Flash 390-2; SMC: 1.16) from Linux Gold Update 3 (Build: 2.1.6720-13; Flash:386-2; SMC:1.14)).

Table 5. Calculation times $T_{\rm CALC}$ corresponding to Fig. 11. $P_{N\,\rm per\,cent}$ is the Nth percentile of MVM calculation times that are completed by the given time. All times given in milliseconds.

$T_{\rm CALC}$ (ms)	One Xeon Phi Pre-update	One Xeon Phi Post-update
Jitter (σ)	0.045	0.037
Min	1.297	1.301
Mean	1.349 ($P_{81\,\rm per\,cent}$)	1.348 ($P_{77\,\rm per\,cent}$)
$P_{99\,\rm per\,cent}$	1.560	1.463
$P_{99.9\,\rm per\,cent}$	1.701	1.522
$P_{99.99\,\rm per\,cent}$	1.749	1.957
$P_{99.999\,\rm per\,cent}$	1.765	2.306
Max	1.813	2.505

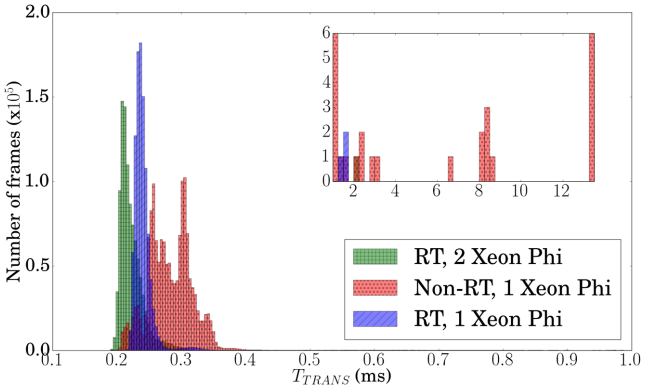


Figure 12. Histogram comparing the transfer time $T_{\rm TRANS}$ for a 80 × 80 sub-aperture system calculated using 10^6 samples. $T_{\rm TRANS}$ is the combined time for transferring data in and out of the Xeon Phi. Blue: single Xeon Phi on an RT Linux kernel; Red: single Xeon Phi on non-RT Linux. Inset shows data from 1 to 13 ms with the number of frames ranging from 0–6 (showing outliers more clearly).

not until $P_{99.99\,\rm per\,cent}$ that we see that the outliers become worse after the update. This is a problem for the performance of the system, it also highlights a larger problem of using hardware accelerators such as the Xeon Phi or GPUs. Neither of these technologies are designed for RT control systems, and any AO RTC system based on hardware accelerators is tied to the development and direction the company developing them decides on. An upgrade that is beneficial to high performance computing (HPC) or gaming community may degrade AO RTC performance. As a result, and since mean and jitter cannot fully characterize hardware for AO applications, it is necessary to analyse the full distribution of frame computation times when comparing or upgrading hardware.

3.2.3 *Variability in data transfer:* T_{TRANS}

The larger outliers seen in $T_{\rm OFFL}$ were not seen in $T_{\rm CALC}$, suggesting that the main cause lies in transfer time $T_{\rm TRANS}$. Fig. 12 shows the distribution of data transfer timings for both a host computer running non-RT and RT Linux system. Table 6 shows the detailed results for the data transfer. It demonstrates that the large outliers seen in the non-RT $T_{\rm OFFL}$ are indeed caused by the transfer of data between the host computer and the accelerator. We see that moving to a RT Linux has reduced the outliers bringing the maximum values from 13.507 ms down to 1.747 ms, a large reduction. It has also suppressed most of the outliers, but not all, and reduced jitter from 0.048 to 0.014 ms. On average, the system spends 15 per cent of

Table 6. Transfer times T_{TRANS} corresponding to Fig. 12. $P_{N\,\mathrm{per\,cent}}$ is the Nth percentile of transfer times that are completed by the given time. All times given in milliseconds.

T_{TRANS} (ms)	One Xeon Phi Non-RT	One Xeon Phi RT
Jitter (σ)	0.048	0.014
Min	0.196	0.204
Mean	0.283 ($P_{81\,\mathrm{per\,cent}}$)	0.239 ($P_{77\,\mathrm{per\,cent}}$)
$P_{99\,\mathrm{per\,cent}}$	0.359	0.313
$P_{99.9\,\mathrm{per\,cent}}$	0.592	0.344
$P_{99.99\,\mathrm{per\,cent}}$	0.752	0.399
$P_{99.999\,\mathrm{per\,cent}}$	2.957	0.647
Max	13.507	1.747

Table 7. Entire AO frame processing times T_{RTCS} corresponding to Fig. 13. $P_{N\,\mathrm{per\,cent}}$ is the Nth percentile of entire AO frame processing times that are completed by the given time. All times given in milliseconds.

T_{RTCS} (ms)	One Xeon Phi non-RT	One Xeon Phi RT	Xeon E5 non-RT	Xeon E5 RT
Jitter (σ)	0.193	0.061	1.5924	0.570
Min	1.920	2.154	6.120	6.717
Mean ($P_{XX\,\mathrm{per\,cent}}$)	2.260 ($P_{56\,\mathrm{per\,cent}}$)	2.320 ($P_{67\,\mathrm{per\,cent}}$)	8.550 ($P_{54\,\mathrm{per\,cent}}$)	8.366 ($P_{53\,\mathrm{per\,cent}}$)
$P_{99\,\mathrm{per\,cent}}$	2.642	2.545	11.058	10.065
$P_{99.9\,\mathrm{per\,cent}}$	2.848	2.921	11.455	10.816
$P_{99.99\,\mathrm{per\,cent}}$	3.179	3.267	16.345	11.362
$P_{99.999\,\mathrm{per\,cent}}$	22.504	3.570	20.727	11.773
Max	103.249	11.759	128.028	12.077

the total offload time transferring data in and out of the Xeon Phi. When adding a second Xeon Phi (data not shown here), the transfer time T_{TRANS} is not reduced by much, even though only half the data needs to be transferred to each Xeon Phi. This transfer happens simultaneously but due to the overheads involved, T_{TRANS} cannot be reduced by a significant amount.

In order to increase robustness, a method for reducing transfer (and calculation) time variability could be devised by using two Xeon Phis performing identical calculations. When the fastest Xeon Phi has finished its calculations, the other is interrupted to be ready to receive slopes from the next frame. There is no simple functionality to interrupt a call to the MKL library running on the Xeon Phi once it has started or to stop the data transfer. This issue has not been investigated in this paper, and it is believed that the next Xeon Phi generation (standalone CPU with no data transfer, see Section 4) will be able to run an RT Linux system therefore removing almost entirely these very high latency events.

3.2.4 *Integration of Xeon Phi into an AO RTC software:* T_{RTCS}

We have shown that the Xeon Phi is able to reduce MVM calculation time for large systems over that of modern CPUs, such as the Xeon E5. In this section, we integrate the Xeon Phi into a complete AO RT control software. We chose to integrate the Xeon Phi into Durham Adaptive optics Real-time Controller (Basden et al. 2010b), which is currently being used on the William Herschel Telescope for the CANARY AO demonstrator. In this section, we run an end-to-end simulation of an AO RTC system using simulated wavefront sensor data. The measured time T_{RTCS} includes WFS pixel processing,

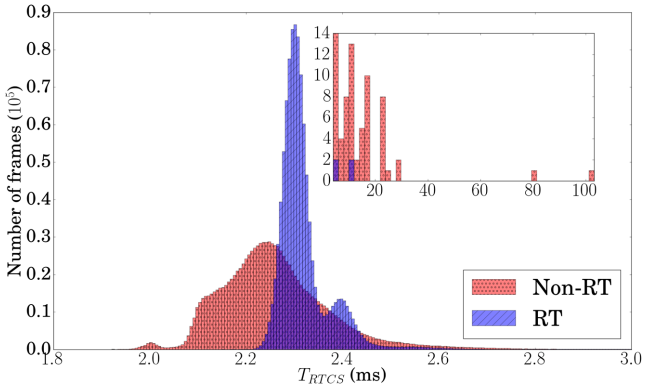


Figure 13. Histogram comparing the entire AO frame processing T_{RTCS} for a 80 × 80 sub-aperture system calculated using 10^6 samples. T_{RTCS} encapsulates MVM calculation time, transfer time, pixel processing, slope computation and any overhead of running the RTC system. Blue: single Xeon Phi on RT Linux kernel. Red: single Xeon Phi on a non-RT Linux. Inset shows data from 4 to 103 ms with number of frames ranging from 0–14 (showing outliers more clearly).

slope calculation, the MVM calculation on the Xeon Phi as well as additional background tasks of the RTC system.

Only a single thread is able to transfer data to the Xeon Phi at one time. The transfer step has a larger amount of overhead when compared the data transfer size. This overhead means that if we pipeline the AO control loop and transfer groups of slopes to the Xeon Phi and split the MVM into multiple smaller MVMs the overall T_{RTCS} is increased. Although pipelining the MVM is possible, it was decided here to perform a single MVM per frame, when all slopes have been calculated. For the next generation of the Xeon Phi, where there is no transfer step, pipelining will be more appropriate.

Fig. 13 shows the comparison between T_{RTCS} running both non-RT and RT Linux using a Xeon Phi to accelerate the MVM. Table 7 shows more detailed results of the RTC processing time, analysing the distributions in terms of minimum and maximum values, jitter, mean and percentiles. It also shows the results for the RTCS running on the Xeon E5 only, without Xeon Phi acceleration. Offloading the MVM to the Xeon Phi and running on an RT Linux system brings the jitter down (i.e. narrower distribution) and reduces the number of outliers, although they are not completely eliminated. The jitter of the whole RTC is reduced from 0.193 ms for a non-RT Linux down to 0.061 ms for an RT Linux, a sizeable reduction. The outliers appear to be far worse than for the standalone tests; this is likely to be due to the fact that the CPU is now stressed with other tasks such as WFS data processing and thread synchronization. As expected, moving to an RT kernel has made the maximum value drop, from 103 to 11.8 ms. The minimum and mean values have however increased slightly on the RT system; this is likely due to how the internals of the RT kernel work allowing a process with raised priority to pre-empt other processes. RT Linux systems do not optimize for overall performance, they optimize for reliability and predictability.

Although this performance would not be sufficient for typical first-light E-ELT instruments (e.g. mean frame time of 2.3 ms), we have demonstrated in this section that incorporating the Xeon Phi into an existing AO RT control software can be done efficiently without investing a significant amount of time and effort and has the potential to improve the RTC performance.

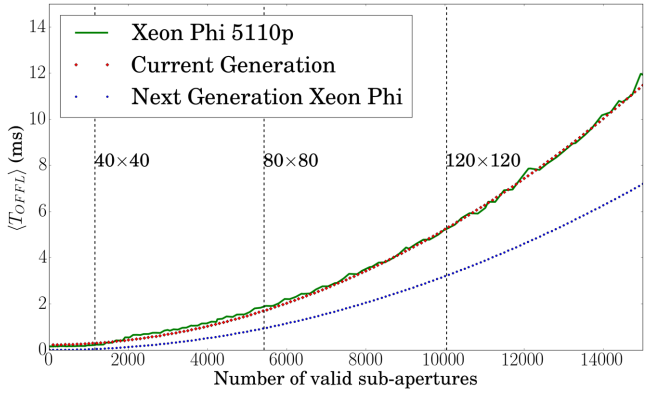


Figure 14. Comparison of mean MVM execution time between the current Xeon Phi (5110P) and the predicted next generation. These projections are based on the specifications found in Table A1 in the appendix. Green: current version; Red: current version based on predictive model; Blue: next generation estimation based on same model.

4 PROSPECTIVE EVOLUTION OF THE XEON PHI

The next generation of Xeon Phi has been announced and will likely come in two variations. The first version is the same as the current version: an accelerator card connecting to a host computer over the PCIe bus. The second variation will be available as a standalone CPU and is the variation that seems to offer the most potential for AO applications. As a standalone CPU it will remove the need to transfer data to the host and back, which causes the large outliers and introduces additional latency. It is expected to support the standard operating systems, including an RT Linux kernel, further removing the amount of calculation jitter.

Some specifications have been disclosed by Intel on the next generation of Xeon Phis, codenamed 'Knights Landing' (Intel 2014), and can be found in Table A1. The next Xeon Phi cores will be based on the Intel Atom CPUs which have a cache of 512 kB each and the total number of cores will be 60+. The clock speed is expected to be ≈1–1.5 GHz which is slower than modern CPUs, making the Xeon Phi likely to suffer the same performance problem when running serial code. The type of CPU used suggests that the L2 cache of the entire system will be 30–35 MB. The size of control matrix for ELT first light instruments is however considerably larger than this (at least 205 MB) and we can safely assume that the MVM will still be memory bandwidth limited. The specifications suggest a memory bandwidth of over 500 GB s^{-1}. In our tests, we found that the actual achievable memory bandwidth was close to half the stated theoretical maximum; we are likely to be able to achieve a memory bandwidth of ≈250 GB s^{-1}.

From these specifications, we can estimate the average performance (see Fig. 14) of the current Xeon Phi and how the next generation might scale as a function of system size. The mean performance of the next generation rivals that of two Xeon Phis of current generation. For an 80 × 80 system $\langle T_{\rm OFFL}\rangle = \langle T_{\rm CALC}\rangle = 0.86$ ms (1.587 ms for the current generation), which is compatible with the 500 Hz frame rate of first-light E-ELT instruments. The large increase in performance is mainly due to the removal of the data transfer step and to higher memory bandwidth. Jitter and outliers are harder to predict. It is reasonable however, to assume that the distribution curve for $T_{\rm CALC}$ will be similar to that of the current generation and that running an RT Linux kernel on the Xeon Phi will further reduce both jitter and the number of outliers. The next Xeon Phi generation, being a standalone CPU, has the potential to offer the performance benefits of the current hardware accelerators (e.g. Xeon Phi, GPUs) while removing the main disadvantages of this technology: the transfer of data.

In this paper, we have only considered the use of a unique control matrix throughout the operation of the AO system. In reality, the control matrix will need to be updated as the observation condition changes; for the E-ELT this is likely to be of the order of minutes. The Xeon Phi offers asynchronous transfer of data which should allow for the matrix to be updated during calculations. It is likely however to interfere with performance. The next generation of Xeon Phi (standalone CPU) should mitigate the impact of control matrix swapping by allowing a new matrix to be uploaded without transferring it over the PCIe bus. Although maybe small, the impact of having the MVM coefficients not in L3 cache needs to be investigated. Transferring a updated control matrix into memory will reduce the memory bandwidth for the MVM calculation. To lessen the impact of this on performance, the updated matrix can be uploaded over several iterations.

5 CONCLUSIONS

In this paper, we have presented a detailed study of the Xeon Phi, a many-core processor, used as compute accelerator for AO RT applications. We investigated performance for the most compute intense part of the RTC: the wavefront reconstruction. Our examination focused on the MVM algorithm, the most commonly used and the most parallelizable of wavefront reconstruction algorithms. This enables us to take full advantage of the high number of cores of the Xeon Phi. We described how AO system size and the number of Xeon Phis impact performance and investigated the main contributors to the time delay splitting between data transfer time and MVM calculation. Finally, we discussed the implementation ease and overall performance of integrating the Xeon Phi into a complete RTC software.

We demonstrated that performance changes gradually over the whole range of control matrix sizes studied, and that performance for a specific AO system can easily be assessed by scaling. We believe that this paper can serve as a guideline for estimating MVM performance for any AO system size using a single or multiple Xeon Phis. A more detailed analysis also showed that mean execution time is rarely sufficient to fully qualify novel hardware (or when updating firmware) in RT applications and that the actual distribution of execution times needs to be analysed in detail. To make the comparison between potential RTC hardware more tractable, we decided to present results in terms of minimum, maximum, mean, deviation (jitter) and percentile of execution time.

Using the Xeon Phi enables a clear improvement in MVM mean calculation time, whether tested as a standalone system or fully integrated into the RTC software. We have shown that moving the host from a non-RT to an RT Linux system can naturally reduce the number and extent of outliers, as well as reduce mean offload times. For a typical 80 × 80 E-ELT first-light SCAO system, mean offload time $\langle T_{\rm OFFL}\rangle \approx 1.587$ ms and 99.999 per cent of the offloads are finished within ≈2.663 ms. However, a number of outliers are still present (most likely due to the fact that the Xeon Phi is running a non-RT micro-Linux) probably making the current generation of this technology only suitable for some AO RT applications [e.g. ground layer adaptive optics (GLAO), multi object adaptive optics (MOAO)] but unsuitable for others [e.g. eXtreme adaptive optics (XAO)].

Sharing calculations between two Xeon Phis allows us to further reduce mean offload time $\langle T_{\rm OFFL} \rangle$. The maximum speedup between one and two Xeon Phis plateaus at around 1.8 for large systems, and the speedup for a typical 80 × 80 E-ELT first-light SCAO system reaches 1.6. In this configuration, the mean offload time $\langle T_{\rm OFFL} \rangle \approx 0.978$ ms and 99.999 per cent of the offloads are finished within ≈2.118 ms. This shows the scalability of a system using multiple Xeon Phis, and it is reasonable to assume that adding more Xeon Phis would further reduce the latency in a similar way.

The Xeon Phi is designed to be used within supercomputers, and the HPC community is generally more focused on data throughput rather than on time-critical processes. We have found that the variability in execution time (increased jitter and outliers) can increase after firmware updates. Using the Xeon Phi as an offload card turns a homogeneous CPU system into a heterogeneous computing environment, which is more complex to programme and to balance work loads efficiently. On the other hand, the theoretical memory bandwidth of the Xeon Phi is very high, which is essential for a bandwidth limited problem such as the MVM. We have shown that about 50 per cent of the theoretical memory bandwidth is achievable, in line with other findings (Fang et al. 2014). In addition, we have shown that the achievable memory bandwidth can offer a good estimate for the mean performance of the Xeon Phi calculating the MVM, and that most of the outliers come from transferring data in and out of the Xeon Phi. The expected next Xeon Phi generation has great potential in being suitable for AO, being an integrated CPU, eliminating the need to transfer data over the PCIe bus, and also offering higher compute power. Both mean RTC performance, jitter and outliers have the potential to be greatly reduced in forthcoming hardware.

ACKNOWLEDGEMENTS

This work is part funded by the Science and Technology Facilities Council (STFC), grant ST/K003569/1 and the Centre For Instrumentation. We gratefully acknowledge support for this research from the UK Engineering and Physical Sciences Research Council, under Grant number EP/L01596X/1.

REFERENCES

Babcock H. W., 1953, PASP, 65, 229
Barr D., Basden A., Dipper N., Schwartz N., Vick A., Schnetler H., 2014, in Marchetti E., Close L. M., Véran J.-P., eds, Proc. SPIE Conf. Ser. Vol. 9148, Adaptive Optics Systems IV. SPIE, Bellingham, p. 91484B
Basden A., Myers R., 2012, MNRAS, 424, 1483
Basden A., Myers R., Butterley T., 2010a, Appl. Opt., 49, G1
Basden A., Gang D., Myers R., Younger E., 2010b, Appl. Opt., 49, 6354
Basden A., Dipper N., Myers R., Younger E., 2012, in Ellerbroek B. L., Marchetti E., Véran J.-P., eds, Proc. SPIE Conf. Ser. Vol. 8447, Adaptive Optics Systems III. SPIE, Bellingham, p. 84472S
Bosma W., Cannon J., Playoust C., 1997, J. Symb. Comput., 24, 235; Computational Algebra and Number Theory (London, 1993)
Bouchez A. H. et al., 2008, in Hubin N., Max C. E., Wizinowich P. L., eds, Proc. SPIE Conf. Ser. Vol. 7015, Adaptive Optics Systems. SPIE, Bellingham, p. 70150Z
Bouchez A. H. et al., 2009, in Heaney J. B., Warren P. G., Tyson R. K., Marshall C. J., Kvamme E. T., Hart M., eds, Proc. SPIE Conf. Ser. Vol. 7439, Astronomical and Space Optical Systems. SPIE, Bellingham, p. 74390H
Boyer C. et al., 2014, in Marchetti E., Close L. M., Véran J.-P., eds, Proc. SPIE Conf. Ser. Vol. 9148, Adaptive Optics Systems IV. SPIE, Bellingham, p. 91480X
Evans C. J. et al., 2011, A&A, 527, A50
Fang J., Sips H., Zhang L., Xu C., Che Y., Varbanescu A. L., 2014, in Klaus-Dieter L., John M., eds, Proc. 5th ACM/SPEC Int. Conf. (ICPE '14) on Performance Engineering. ACM, New York, NY, p. 137
Fedrigo E., Donaldson R., Soenke C., Myers R., Goodsell S., Geng D., Saunter C., Dipper N., 2006, in Ellerbroek B. L., Calia D. B., eds, Proc. SPIE Conf. Ser. Vol. 6272, Advances in Adaptive Optics II. SPIE, Bellingham, p. 627210
Fusco T., Thatte N., Meimon S., Tecza M., Clarke F., Swinbank M., 2010, in Ellerbroek B. L., Hart M., Hubin N., Wizinowich P. L., eds, Proc. SPIE Conf. Ser. Vol. 7736, Adaptive Optics Systems II. SPIE, Bellingham, p. 773633
Gratadour D., Sevin A., Brulé J., Gendron E., Rousset G., 2012, in Ellerbroek B. L., Marchetti E., Véran J.-P., eds, Proc. SPIE Conf. Ser. Vol. 8447, Adaptive Optics Systems III. SPIE, Bellingham, p. 84475R
Intel, 2014, What public disclosures has intel made about knights landing? Available at: https://software.intel.com
McCalpin J. D., 1991-2007, Tech. rep., Stream: Sustainable Memory Bandwidth in High Performance Computers. Univ. Virginia, Available at: http://www.cs.virginia.edu/stream/
Mauch S., Reger J., Reinlein C., Appelfelder M., Goy M., Beckert E., Tünnermann A., 2014, in Bifano T. G., Kubby J., Gigan S., eds, Proc. SPIE Conf. Ser. Vol. 8978, MEMS Adaptive Optics VIII. SPIE, Bellingham, p. 897802
Paufique J. et al., 2012, in Ellerbroek B. L., Marchetti E., Véran J.-P., eds, Proc. SPIE Conf. Ser. Vol. 8447, Adaptive Optics Systems III. SPIE, Bellingham, p. 844738
Pazder J., Bauman B., Dillon D., Fletcher M., Lacoursière J., Reshetov V., 2012, in Navarro R., Cunningham C. R., Prieto E., eds, Proc. SPIE Conf. Ser. Vol. 8450, Modern Technologies in Space- and Ground-based Telescopes and Instrumentation II. SPIE, Bellingham, p. 845058
Poyneer L., Gavel D., Brase J., 2002, J. Opt. Soc. Am. A, 19, 2100
Puech M., Rosati P., Toft S., Cimatti A., Neichel B., Fusco T., 2010, MNRAS, 402, 903
Reguly I. Z., László E., Mudalige G. R., Giles M. B., 2014, in Pavan B., Minyi G., Zhiyi H., eds, Proc. Programming Models and Applications (PMAM'14) on Multicores and Manycores. ACM, New York, NY, p. 39
Rosensteiner M., 2012, J. Opt. Soc. Am. A, 29, 2328
Sauvage J.-F. et al., 2010, in Ellerbroek B. L., Hart M., Hubin N., Wizinowich P. L., eds, Proc. SPIE Conf. Ser. Vol. 7736, Adaptive Optics Systems II. SPIE, Bellingham, p. 77360F
Sevin A., Perret D., Gratadour D., Lainé M., Brulé J., Le Ruyet B., 2014, in Marchetti E., Close L. M., Véran J.-P., eds, Proc. SPIE Conf. Ser., Vol. 9148, Adaptive Optics Systems IV. SPIE, Bellingham, p. 91482G
Smith M., Kerley D., Herriot G., Véran J.-P., 2014, in Marchetti E., Close L. M., Véran J.-P., eds, Proc. SPIE Conf. Ser., Vol. 9148, Adaptive Optics Systems IV. SPIE, Bellingham, p. 91484K
Véran J.-P. et al., 2014, in Marchetti E., Close L. M., Véran J.-P., eds, Proc. SPIE Conf. Ser., Vol. 9148, Adaptive Optics Systems IV. SPIE, Bellingham, p. 91482F
Vérinaud C. et al., 2010, in Ellerbroek B. L., Hart M., Hubin N., Wizinowich P. L., eds, Proc. SPIE Conf. Ser. Vol. 7736, Adaptive Optics Systems II. SPIE, Bellingham, p. 77361N
Wang L., 2013, in Esposito S., Fini L., eds, Proc. Third AO4ELT Conf., p. 17, Available at: http://ao4elt3.sciencesconf.org/
Wang L., Ellerbroek B., 2012, in Ellerbroek B. L., Marchetti E., Véran J.-P., eds, Proc. SPIE Conf. Ser. Vol. 8447, Adaptive Optics Systems III. SPIE, Bellingham, p. 844723
Zhang H., Ljusic Z., Hovey G., Véran J.-P., Herriot G., Dumas M., 2012, in Ellerbroek B. L., Marchetti E., Véran J.-P., eds, Proc. SPIE Conf. Ser. Vol. 8447, Adaptive Optics Systems III. SPIE, Bellingham, p. 84472E

APPENDIX A: HARDWARE SPECIFICATIONS

Table A1. Specifications of hardware being used as well as announced next generation Xeon Phi, two versions are planned one offload via PCIe and one as a standalone CPU.

	Xeon	Xeon Phi	Xeon Phi
Processor	E5-2650	5110p	Knights landing
Release year	2012	2012	2015–2016
#Cores	32	60	60–72
Clock speed	1.20 GHz	1.053 GHz	–
L2 Cache	20 MB	30 MB	–
Memory type	DDR3	GDDR5	DDR4
Memory bandwidth	51.2 GB s^{-1}	320 GB s^{-1}	500+ GB s^{-1}
PCIe (# lanes)	N/A	2.0 ($\times$16)	N/A/ 3.0 ($\times$36)[a]

Note. [a]Knights landing can be purchases as either a co-processing card or standalone CPU.

This paper has been typeset from a TEX/LATEX file prepared by the author.

A.2 Calculating the valid number of sub-apertures

Due to the shape of the telescope pupil (and in particular the presence of spider and of a central obscuration), not all sub-apertures are fully illuminated. Different telescopes have different central obstructions meaning a slightly different number of valid sub-apertures for a given AO system size.

In the tests run in this thesis we are considering the European-Extremely Large Telescope, while ignoring the presence of spiders. For example, a system that has 74×74 sub-aperture SH-WFS will have a total of 5476 sub-apertures while approximately 4000 valid sub-apertures. A similar scaling between total number of sub-apertures and valid number of sub-apertures is used throughout this document.

A.3 Units in computing

In computers, information is stored as 1s and 0s. A single 1 or 0 is a single bit, the smallest memory unit in a computer. To represent more complex numbers, 1s and 0s are put together to create the binary representation of numbers. Table A.1 shows the list of numbers that can be represented with four bits. Four bits are able to represent 0-15 $(2^4 - 1)$.

Table A.1: A subset of the numbers representable with four bits

0	1	2	3	4	5	...	15
0000	0001	0010	0011	0100	0101	...	1111

Typically in computing, the smallest block of memory (8 bits) is called a Byte, and is used to represent the values of 0-255 $(2^8 - 1)$.

Integers and Floating point numbers are typically represented by four Bytes (32 bits).

To simplify large values in many areas, the prefixs or k, M, G are used to represent values of 10^3, 10^6 and 10^9. In computing this is no different. Due to the base 2 of the memory structure, two separate systems are used: ki, Mi, Gi refers to multiples of 1024 (2^{10}), and the standard k, M, G for multiples of 1000 (10^3). Table A.2 shows the values these prefixes represent. In this thesis, we will typically use multiples of 1000.

Table A.2: Orders of magnitudes of memory

-	Bytes	-	Bytes
1 kB	1000	1 kiB	1024
1 MB	1,000,000	1 MiB	1,048,576
1 GB	1,000,000,000	1 GiB	1,073,741,824

A.4 Data transfer units

The units used for the transfer of data (i.e moving a file from one computer to another) is typically bits per second ($bits^{-1}$) rather than Bytes per second (Bs^{-1}).

In this thesis, we discuss the transfer rate of 1 Gigabit Ethernet and 10 Gigabit Ethernet. To distinguish between the standard of Gigabit Ethernet and actual measured rates, we use the abbreviation GbE for the standard and Gbit s^{-1} for the measured transfer rates.